Analysis and Design of Authentication and Encryption Algorithms for Secure Cloud Systems

by

Bo Zhu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Along with the fast growth of networks and mobile devices, cloud computing has become one of the most attractive and effective technologies and business solutions nowadays. Increasing numbers of organizations and customers are migrating their businesses and data to the cloud due to the flexibility and cost-efficiency of cloud systems. Preventing unauthorized access of sensitive data in the cloud has been one of the biggest challenges when designing a secure cloud system, and it strongly relies on the chosen authentication and encryption algorithms for providing authenticity and confidentiality, respectively. This thesis investigates various aspects of authentication and encryption algorithms for securing cloud systems, including authenticated encryption modes of operation, block ciphers, password hashing algorithms, and password-less/two-factor authentication mechanisms.

Improving Authenticated Encryption Modes. The Galois/Counter Mode (GCM) is an authenticated encryption mode of operation for block ciphers. It has been widely adopted by many network standards and protocols that protect the security of cloud communications, such as TLS v1.2, IEEE 802.1AE and IPsec. Iwata *et al.* recently found a flaw in GCM's original proofs for non-96-bit nonce cases, and then presented new security bounds for GCM. The new bounds imply that the success probabilities of adversaries for attacking GCM are much larger than the originally expected ones. We propose a simple change to repair GCM. When applied, it will improve the security bounds by a factor of about 2^{20} while maintaining most of the original proofs.

Analyzing Polynomial-Based Message Authentication Codes. We investigate attacks on polynomial-based message authentication code (MAC) schemes including the one adopted in GCM. We demonstrate that constructing successful forgeries of these MAC schemes does not necessarily require hash collisions. This discovery removes certain restrictions in the attacks previously proposed by Procter and Cid. Moreover, utilizing a special design of GCM for processing non-96-bit nonces, we turn these forgery attacks into birthday attacks, which will significantly increase their success probabilities. Therefore, by considering the birthday attacks and the security proof flaw found by Iwata *et al.*, cloud system designers should avoid using GCM with non-96-bit nonces if they do not revise the design of GCM.

Analyzing Block Ciphers. We propose a new framework for analyzing symmetric-key ciphers by guessing intermediate states to divide ciphers into small components. This framework is suitable for lightweight ciphers with simple key schedules and block sizes smaller than key lengths. Using this framework, we design new attacks on the block cipher family KATAN. These attacks can recover the master keys of 175-round KATAN32, 130-round KATAN48 and 112-round KATAN64 faster than exhaustive search, and thus reach

many more rounds than the existing attacks. We also provide new attacks on 115-round KATAN32 and 100-round KATAN48 in order to demonstrate that this new kind of attack can be more time-efficient and memory-efficient than the existing ones.

Designing Password Hashing Algorithms. Securely storing passwords and deriving cryptographic keys from passwords are also crucial for most secure cloud system designs. However, choices of well-studied password hashing algorithms are extremely limited, as their security requirements and design principles are different from common cryptographic primitives. We propose two practical password hashing algorithms, PLECO and PLECTRON. They are built upon well-understood cryptographic algorithms, and combine the advantages of symmetric-key and asymmetric-key primitives. By employing the Rabin cryptosystem, we prove that the one-wayness of PLECO is at least as strong as the hard problem of integer factorization. In addition, both password hashing algorithms are designed to be sequential memory-hard, in order to thwart large-scale password searching using parallel hardware, such as GPUs, FPGAs, and ASICs.

Designing Password-less/Two-Factor Authentication Mechanisms. Motivated by a number of recent industry initiatives, we propose LOXIN, an innovative solution for password-less authentication for cloud systems and web applications. LOXIN aims to improve on passwords with respect to both usability and security. It utilizes push message services for mobile devices to initiate authentication transactions based on asymmetric-key cryptography, and enables users to access multiple services by using pre-owned identities, such as email addresses. In particular, the LOXIN server cannot generate users' authentication credentials, thereby eliminating the potential risk of credential leakage if the LOXIN server gets compromised. Furthermore, LOXIN is fully compatible with existing password-based authentication systems, and thus can serve as a two-factor authentication mechanism.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude towards my advisor Professor Guang Gong for her excellent guidance and tremendous support during the past four years. Her enthusiasm for research and optimistic attitude have always inspired and motivated me, and her extensive knowledge and experience have been the biggest help and resource for my studies. The last four years have been an enjoyable and unforgettable time that has given me great academic and personal development, thanks to the role model that Professor Gong has provided as a successful person, researcher, and professor.

I would also like to express my sincerest appreciation to Professor Kui Ren from the University at Buffalo, The State University of New York for serving as my external examiner and providing many valuable suggestions. My deepest gratitude also goes to my thesis examining committee members, Professor Catherine Gebotys, Professor Kshirasagar Naik, Professor Edlyn Teske-Wilson, and Professor Mahesh Tripunitara, for their instrumental comments and their time on this thesis. I am deeply honored and blessed to have been guided by them. The thesis would not have been possible without their assistance.

I am particularly grateful for the knowledge, inspiration, and encouragement that I received from Professor Kefei Chen and Professor Xuejia Lai at Shanghai Jiao Tong University during my Master's study. It was they who motivated me and led me to pursue my interests in cryptography and security.

I am also indebted to my friends and colleagues at the University of Waterloo for their enormous support and inspiring suggestions: Professor Mark Aagaard, Professor Honggang Hu, Dr. Xinxin Fan, Dr. Fei Huo, Dr. Zhijun Li, Dr. Yiyuan Luo, Dr. Kalikinkar Mandal, Dr. Yin Tan, Dr. Yang Yang, Yao Chen, Teng Wu, Gangqiang Yang, and Shasha Zhu. I am so lucky to be one of the members of the Communication Security (ComSec) Lab, which has always been a pleasant and stimulating research atmosphere.

Last but not least, I want to thank my parents for their unconditional and endless love, trust, support and understanding. Thanks go to my father for guiding me into the fascinating world of mathematics and physics when I was young. Thanks are due to my mother for her care that has always kept me full of courage. None of my work would have been possible without them.

To my parents
To my family

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AEAD	Authenticated encryption with associated data
ASIC	Application-specific integrated circuit
CA	Certificate authority
FPGA	Field-programmable gate array
FSR	Feedback shift register
GPU	Graphics processing unit
IDP	Identity provider
IV	Initial vector
KDF	Key derivation function
LFSR	Linear feedback shift register
LSB	Least significant bit
MAC	Message authentication code
MITM	Meet-in-the-middle
MSB	Most significant bit
PMS	Push message service
XOR	Exclusive-or

List of Notations

0^n	An n-bit all-zero binary string, where $n \geq 0$ (0^n denotes an empty string if $n = 0$)
$s_1 \| \| s_2$	Concatenating two binary strings s_1 and s_2
$\|\mathcal{S}\|$	The cardinality of a set $\mathcal{S}$
$\mathsf{int}(s)$	Converting a binary string s to a non-negative integer, where the endianness depends on the context
$\mathsf{len}(s)$	The bit-length of the binary string s
$\mathsf{lsb}_n(s)$	The rightmost n bits of s
$\mathsf{msb}_n(s)$	The leftmost n bits of s
$\mathsf{size}(x)$	The number of bits in the shortest binary representation of the given positive integer x, e.g., $\mathsf{size}(256) = 9$ and $\mathsf{size}(255) = 8$
$\mathsf{str}_n(x)$	Converting a non-negative integer x to a binary string, and prepending/appending zero bits to the string in order to achieve a total length of n bits, where the endianness depends on the context

Chapter 1

Introduction

Cloud computing has quickly gained its popularity over traditional software and hardware models over the last decade. While companies and customers are moving their data and businesses to the cloud, people raise more and more concerns about the security and privacy of cloud systems. Protecting customers' data and businesses in the cloud is vital to all cloud system designers and service providers. Our work in this thesis aims to provide insights into authentication and encryption algorithms used in cloud systems.

In this chapter, Section 1.1 briefly introduces cloud systems. Section 1.2 gives the background knowledge about various kinds of authentication and encryption algorithms for securing cloud systems. Subsequently, our motivations and related work are discussed in Section 1.3. Finally, our contributions are summarized in Section 1.4.

1.1 A Brief Introduction to Cloud Systems

Along with the advances of network technologies and mobile devices, nowadays organizations and users tend to outsource their computation and storage needs to various cloud services. The vendors of cloud services or cloud systems can provide more cost-efficient and scalable resources than traditional systems due to resource pooling and economies of scale.

The major resources that cloud systems may provide include:

Computation. Customers may outsource their computation-intensive tasks to vendors. For example, by a similar cost one may choose to rent one hundred cloud servers to

finish a task in one hour, rather than use a single server that continuously runs the same task for one hundred hours.

Storage. Vendors provide on-demand capacities for file storage and databases that can be accessed by customers from anywhere.

Network. Customers need to deliver contents to end users reliably and quickly. Such services include *content delivery networks* (CDNs) that cache data on hundreds of global servers and deliver the data to users from the nearest location, and *push message services* that provide channels for sending messages to mobile device users.

The term *cloud computing* has been defined by the National Institute of Standards and Technology (NIST) as "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [87]. As in the description given by NIST, cloud systems provide many benefits compared with traditional on-premise systems, such as:

Scalability. Computation, storage and network capacities can be provisioned in real time as requested, and they can easily scale up or scale down when requirements change. Consequently, customers can avoid guessing capacities and purchasing physical machines beforehand that usually end up with limited capacities or idle resources.

Low-cost. The customers of cloud systems can benefit from lower cost per unit for resources compared with traditional systems, because the usage from thousands of customers may be aggregated to achieve economies of scale. Furthermore, on-demand cloud systems usually run in a pay-as-you-go model, and thus customers only need to pay for the resources they have used.

Reliability. Computational tasks and data storage can be allocated across different regions or even different vendors in order to provide high availabilities, since customers' overall businesses will not be affected if an unforeseen incident like power outage happens in one location.

Accessibility. The resources in cloud systems are commonly accessible from anywhere as long as the Internet is available. Customers usually connect to cloud systems through thin-client software like web browsers or via web-based application programming interfaces (APIs).

Although cloud computing is a promising technical and business solution, it did not originate from the emergence of a specific new technology. Cloud systems are powered by the combination of advances of pre-existing technologies, such as high-capacity disk storage, high-speed networks, powerful processors, and especially virtualization technologies. Resources are pooled together and virtualized in order to improve utilization and provide a unified interface to customers. A single machine may serve its resources to multiple customers, while customers' resources may also spread over different servers. However, different customers should be isolated from each other, and every customer sees requested resources as a whole and does not need to care about the underlying implementations, due to the advanced virtualization technologies. By different levels of virtualization, cloud systems are mainly delivered in the following three models.

Infrastructure-as-a-Service (IaaS) allows customers to request and assemble the basic units of virtualized computation, storage, and network capacities, and then run arbitrary operating systems and software on them. Customers have flexible control over operating systems, software, and allocated resources, but they may need more human-power to customize and manage their services. Amazon Elastic Compute Cloud (EC2) [6] and Google Compute Engine (GCE) [56] are two examples of IaaS.

Platform-as-a-Service (PaaS) provides pre-configured environments that customers can deploy their applications to. The environments are designed for specific programming languages and databases, and customers use tools provided by vendors to deploy applications and adjust environment settings. Although customers have limited control over underlying resources and environments, they release the heavy burden of maintaining operating systems and infrastructures to vendors, and can focus on their core businesses. The examples of PaaS include Google App Engine (GAE) [55] and AWS Elastic Beanstalk [7].

Software-as-a-Service (SaaS) directly presents the software of a specific use case, such as accounting, collaboration, management, and analysis to end users. In this case, vendors control all the underlying software and hardware, and customers access the provided software via web-based interfaces or certain thin clients. Two examples of SaaS are Google Apps for Work [57] and Salesforce Customer Relationship Management (CRM) solutions [108].

Figure 1.1 gives an illustration of the different architectures of IaaS, PaaS and SaaS, where PaaS might be built upon IaaS, and SaaS may be based on PaaS.

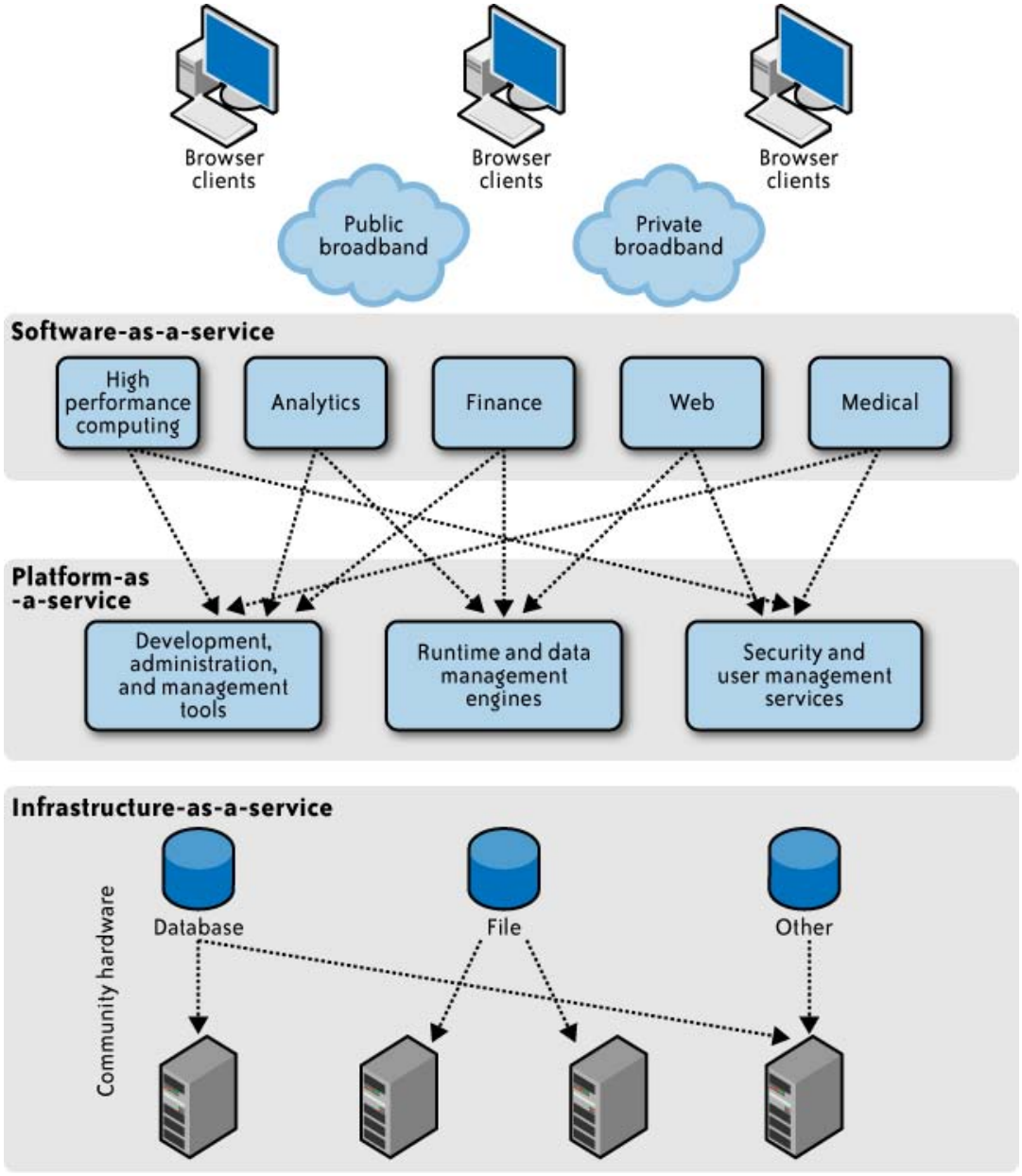

Figure 1.1: Architectures of different models of cloud systems [82].

1.2 Authentication and Encryption for Cloud Systems

Despite the attractive features provided by cloud systems, 74% of IT executives do not intend to migrate their infrastructures or services to cloud systems due to security concerns, according to the surveys [73, 118]. Customers will certainly raise concerns about the privacy and data security of cloud systems, because their data and computation results are stored on shared cloud servers, and may be transmitted through open network connections [103]. In this section, we describe certain critical components in secure cloud systems, such as storage/communication encryption and authentication as well as entity authentication.

1.2.1 Fundamental Security Definitions

In order to build a secure cloud system, similar as other information systems, it requires many important security properties including but not limited to:

Confidentiality implies that only intended parties can read the protected information. Information leakage is an example of violating confidentiality. Data stored in and transmitted to cloud systems may be encrypted to protect confidentiality.

Authenticity refers to that messages, transactions, and documents are assured to be genuine, i.e., created by claimed parties and unaltered by someone else. Please note that authenticity automatically implies *integrity*, where integrity means that data has not been modified in an unauthorized manner.

Availability means that data should be available when it is needed. Availability is vital in cloud systems since customers' businesses may depend on data stored on external cloud servers. For example, *Denial-of-service* (DoS) attacks specifically attempt to affect availability.

This thesis focuses on the confidentiality and authenticity of cloud systems, which are commonly protected by encryption and authentication algorithms, respectively.

1.2.2 Authentication and Encryption for Cloud Communications and Storage

In cloud systems, a large portion of customers' data may be stored and kept by cloud system providers. Customers also need to transfer data back and forth with cloud systems

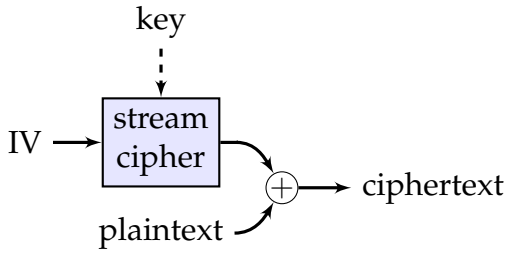

Figure 1.2: Encryption process of stream ciphers.

through the Internet. Therefore, protecting the authenticity and confidentiality of the data in cloud storage and communications is extremely important.

Before sent to the cloud, data can be obscured (i.e., *encrypted*) by various encryption algorithms (also known as *ciphers*), e.g., the Advanced Encryption Standard (AES) [39] or Triple-DES [112], such that only the parities sharing the secret decryption keys can recover the original data. AES and Tripple-DES are both *symmetric-key* ciphers, i.e., encryption keys are the same as decryption keys, and message senders and receivers have to share the same keys in order to perform decryption. For *asymmetric-key* ciphers, encryption and decryptions keys are different. Asymmetric-key algorithms are also called *public-key* algorithms, since one of the two keys is usually public.

The encrypted data can also be transmitted or saved along with a *message authentication code* (MAC) tag that is computed based on the data and a secret authentication key. Consequently, adversaries, without knowing the authentication key, cannot easily create a valid MAC tag mapping to altered data. Legitimate customers can later recompute the MAC tag to verify the integrity and authenticity of the received/retrieved data. CBC-MAC [17] and HMAC [77] are both algorithms for computing MAC tags.

As an example, the popular protocol Transport Layer Security (TLS) [40], commonly used for protecting communications for cloud services, employs various ciphers and MAC algorithms to encrypt and authenticate data.

Stream Ciphers and Block Ciphers

There are two major types of symmetric-key cipher designs: *stream ciphers* that produce (binary) pseudorandom streams (or sequences) to mask plaintexts, and *block ciphers* that take fixed-length blocks of plaintexts as input and output ciphertext blocks with the same length.

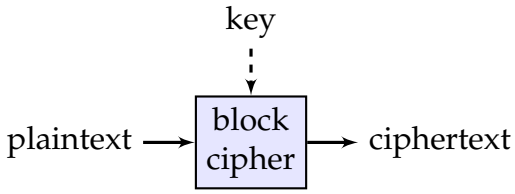

Figure 1.3: Encryption process of block ciphers.

Stream ciphers are usually built on feedback shift registers (FSRs). Stream ciphers first initialize by mixing a master key and an initial vector (IV), then output a pseudorandom sequence, and finally XOR (exclusive-or) the sequence with the plaintext to perform encryption. The master keys are the credentials that should be kept securely, and IVs are (public) random numbers that make the output sequences unique. The encryption process of stream ciphers is illustrated in Figure 1.2. For a fixed secret key, the IV should be changed if the stream cipher algorithm is used to generate a new sequence for encryption. If the IV is repeated, the output stream will be the same as before, and thus may leak information about plaintexts by XORing two streams of ciphertexts together. Two examples of stream ciphers are RC4 [76] and Salsa20 [22].

Block ciphers are designed from another point of view. They only take plaintext blocks with fixed lengths, and output ciphertexts with the same lengths as plaintexts. Block ciphers should be designed to be invertible, i.e., permutations, such that a ciphertext can be unambiguously converted back to its corresponding plaintext. For a plaintext longer than the input size of a block cipher (i.e., *block size*), the plaintext has to be segmented into blocks in order to be encrypted. The model of block ciphers is given in Figure 1.3. AES and Triple-DES are both block ciphers.

Block ciphers alone cannot be used to perfectly protect confidentiality, because plaintext blocks with the same content will produce an identical encrypted block. Consequently, a long ciphertext will reveal certain patterns/distributions of the plaintext. Therefore, block ciphers are usually required to be used with *modes of operation*, such as the CTR [80] and CBC [16] modes, in order to secure long messages. Especially, block ciphers used with some *authenticated encryption modes of operation*, e.g., the CCM [49] and GCM [85] modes, can produce MAC tags along with ciphertexts. This is convenient and efficient for practical applications.

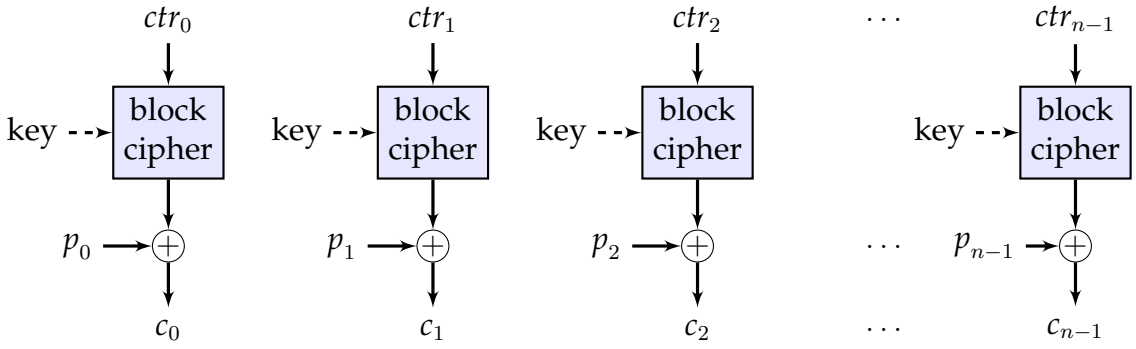

Figure 1.4: Encryption process of block ciphers with the CTR mode.

Modes of Operation for Block Ciphers

There are many popular choices for modes of operation, such as the CTR, CBC, CFB and OFB modes [91]. Here we explain only the CTR (counter) mode as a simple example. The operation of the CTR mode is similar to stream ciphers, i.e., generating a pseudorandom stream and XORing the stream with plaintexts for encryption. The encryption process of block ciphers with the CTR mode is illustrated in Figure 1.4, where p_i and c_i denote plaintexts and ciphertexts, respectively. The counters ctr_i are generated from a given initial counter ctr_0 by using an *incrementing function*. The incrementing function can be applied to either a whole block or a portion of the counter, and the standard incrementing function is simply a modular increment-by-one operation, i.e., $x + 1 \bmod 2^m$, where m denotes the number of bits (i.e., *bit-length*) of the counter portion [48].

The CTR mode can be combined with other MAC algorithms to compute MAC tags when performing encryption. For example, the authenticated encryption mode CCM is a combination of the CTR mode and the CBC-MAC algorithm that provide encryption and authentication, respectively. The Galois/Counter Mode (GCM) is another mode of operation designed based on the CTR mode [104], which possesses many excellent features. GCM is an *authenticated encryption with associated data* (AEAD) mode for block ciphers, which means it will produce a MAC tag simultaneously with the encryption result, and especially the MAC tag can also protect the authenticity of a piece of public information associated with the plaintext (i.e., *associated data*), e.g., the headers of TCP or IP packets. The computations of GCM can be done in parallel (unlike CCM), and only small portions need to be recomputed if one block of input is changed. GCM is included in NSA Suite B Cryptography [92], and has been widely adopted by many standards and protocols, such as TLS v1.2 [40], IEEE 802.1AE [63] and IPsec [119]. A detailed description of GCM is

given in Section 2.1.1.

Attack Goals and Models for Ciphers and Modes of Operation

The attack goals and models that we discuss here are based on Kerckhoffs' principle: The detailed specifications of authentication and encryption algorithms are open to public, and systems' confidentiality and authenticity depend only on the used secret keys that should be randomly chosen. We believe such open designs are more secure and reliable than private ones, because potential weaknesses would be quickly discovered and fixed by the public audience once the algorithm is open to the public in its initial design phase. For a private security algorithm, it would be painful to recover losses if the algorithm is found flawed in production environments. There are various private encryption algorithms that got quickly broken after their detailed designs were leaked, such as A5/1 used in the GSM networks [26], and GMR-1/GMR-2 used for satellite phones' encryption [45].

To attack an encryption algorithm, adversaries' goals may include one of the following:

Plaintext recovery. A (part of) plaintext is recovered from a specific ciphertext.

Key recovery. If adversaries obtain the used secret key, they can easily decrypt ciphertexts to get plaintexts.

To attack an authentication algorithm, except recovering secret keys, adversaries may attempt to achieve:

MAC forgery. Adversaries construct at least one valid message-tag pair that are not produced by legitimate users, where the message is the input (except the authentication key) to the authentication algorithm, e.g., associated data and/or ciphertexts for GCM. This is also called *existential forgery*.

Adversaries can obviously search through the whole space of potential secret keys, plaintexts, or MAC tags, and perform certain simple verification, in order to identify the correct/valid one. This is called *brute-force search* or *exhaustive search*. Therefore, for any encryption or authentication algorithm, its key space must be large enough such that exhaustive search of secret keys is computationally infeasible, e.g., 2^{128} possible choices. The space of plaintexts or MAC tags should also be large enough, depending on different application scenarios. In addition, encryption and authentication algorithms should not have

any serious weaknesses that allow adversaries to recover plaintexts/keys or forge MAC tags faster than exhaustive search.

For analyzing authentication and encryption algorithms, researchers usually use the following well-defined attack models. Each of them stands for a specific level of the capabilities of attackers.

Known-ciphertext attacks. Adversaries only have certain knowledge of existing ciphertexts. For example, encrypted messages are wiretapped by adversaries, but adversaries do not know their corresponding plaintexts. From the known ciphertexts, adversaries may try to recover secret keys or plaintexts.

Known-plaintext or known-message attacks. Adversaries have obtained certain existing plaintext-ciphertext or message-tag pairs, and use such information to compute the secret keys, recover plaintexts from new ciphertexts, or forge MAC tags.

Chosen-plaintext or chosen-message attacks. Adversaries can choose or manipulate the input to the encryption or authentication algorithm. For example, attackers are able to instruct legitimate users to transmit specific messages and then eavesdrop the encrypted messages, which is considered to be a common attack scenario. The *differential attacks* on the Data Encryption Standard (DES) [25] are chosen-plaintext attacks.

Chosen-ciphertext attacks. Adversaries instruct legitimate users to faithfully decrypt any ciphertexts queried by adversaries. In this case, the attack goal is to decrypt other ciphertexts that have not been directly queried.

The above four attack models are listed in the order from the weakest to the strongest. Therefore, for an encryption algorithm, resisting known-ciphertext attacks is the basic requirement, while resisting chosen-ciphertext attacks is the strongest criterion.

There are many other attack models such as *distinguishing attacks*, *related-key attacks* and *side-channel attacks*. Distinguishing attacks aim to find certain non-random behaviors of the algorithms. Related-key attacks target the situations that several master keys used in encryption or authentication algorithms are related, e.g., having a linear relationship. Side-channel attacks are more practical and assume attackers can get physical information such as power consumption, timing, or electrical leakage from circuits. Adversaries could certainly utilize such information for practical attacks. Several other attack models and methods have been discussed in [121, 127].

1.2.3 Entity Authentication for Cloud Systems

As described by Bellare and Rogaway in [18], *entity authentication* is the procedure employed in a distributed system by a communication party to confirm the claimed identity of (i.e., *authenticate*) another party. Entity authentication is extremely important in cloud systems, because in the communications of distributed cloud systems, service providers need to ensure that their customers must not be impersonated by any attackers on the Internet, in order to guarantee the security of customers' data and resources.

One of the most common methods for performing entity authentication is to ask customers to provide a credential, e.g., a password, and then service providers can compare the credential with the one stored in their databases. During the entity authentication process, e.g., logging in to a web-based cloud service, service providers may require a second piece of information customers possess, such as a passcode dynamically generated by an RSA SecurID [106] token, which is called *two-factor authentication*.

Password-Based Authentication

Password-based authentication has been widely deployed in practice due to its simplicity and efficiency. Typically, a user pre-shares a password with a cloud service provider. When the user wants to authenticate himself/herself to the cloud service provider, he/she sends the shared password to the service provider for verification, and the service provider simply compares the received password with the one stored in the database for confirmation. There are two fundamental concerns with this simple password-based authentication mechanism: 1) Users routinely pick low-entropy passwords which are particularly subject to dictionary attacks or brute-force search [35]; 2) A device or server storing a large number of passwords is constantly a target for attackers, and how to store passwords securely and minimize damages if the device or server has been breached is non-trivial.

As an effective countermeasure, all passwords should be obscured together with user-specific high-entropy data (i.e., *salts*) by applying a computation/memory-heavy one-way function, namely *password hashing*, before storing them in the device or server. During the process of entity authentication, the password submitted by the user is processed by the same password hashing algorithm again and then the hashing result (i.e., *hash tag*) is compared with the one stored in the database. In this way, even if hash tags and salts are leaked to attackers, they cannot simply recover users' password by exhaustive search or checking pre-computed look-up tables. In addition, the computation/memory-heavy property would make it more economically difficult for attackers to build efficient

hardware for searching passwords, and thus thwart brute-force attacks to a certain degree. Examples of password hashing algorithms include PBKDF2 [70] and bcrypt [101].

Another important use case of password hashing algorithms is serving as *key derivation functions* (KDFs). KDFs are pseudorandom functions that are used to derive cryptographic keys from certain master or long-term credentials such as passwords. One of the reasons why we need KDFs is that the master credentials like passwords are usually alphanumeric combinations, but cryptographic keys require random, fix-length binary strings. The other reason is that passwords may have low entropies and are vulnerable to brute-force search, so they need to be processed by KDFs for security enhancement [72]. In the setting of cloud systems, customers may first generate a secret key by applying a password hashing algorithm to their passwords, and then use the secret key to encrypt and authenticate their data before sending it to cloud service providers.

Two-Factor Authentication

To further enhance the security of password-based authentication, a promising solution is to deploy a technology called *two-factor*, *second-factor* or *multi-factor authentication*, in which a user is required to provide additional authentication information besides passwords. The second piece of information may be what customers possess such as magnetic banking cards or USB tokens with built-in credentials, or certain biological information such as fingerprints or iris scans. The adoption of two-factor mechanisms makes it more difficult for attackers to bypass the entity authentication of cloud systems, because even if attackers could guess a customer's password correctly, they still need to acquire the specific second piece of information for authentication.

One of the popular two-factor solutions in cloud systems is to require a short passcode generated by a physical token such as RSA SecurID or a software application as Google Authenticator [64]. The technical background of RSA SecurID and Google Authenticator is that the hardware token or software application shares a high-entropy credential with its corresponding authentication server, and a passcode is dynamically generated by applying a pseudorandom sequence generator to the credential together with certain timing information, e.g., the passcode is regenerated every minute.

Attack Goals and Models for Entity Authentication

The goals of adversaries to attack entity authentication mechanisms include:

Impersonation. An adversary convinces a communication partner of a fake identity claimed by the adversary. For example, a malicious user *Mallory* tries to convince *Bob* that she is *Alice* and wants to access *Alice*'s files.

Key (password or credential) recovery is also a goal for adversaries, because if adversaries obtain any passwords or master credentials used for entity authentication, they may be able to impersonate the customers relying on the specific passwords or credentials, by executing the entity authentication protocols or mechanisms as the customers.

There are many attack models relevant to entity authentication mechanisms, including:

Replay attacks. An adversary, say *Mallory*, first eavesdrops and records communications between two parties, *Alice* and *Bob*, in a distributed system. When *Mallory* performs entity authentication with *Bob*, *Mallory* resends *Alice*'s old messages to *Bob*, and lets *Bob* think that he is actually communicating with *Alice*.

Man-in-the-middle attacks. A node in the middle of a communication channel between *Alice* and *Bob*, e.g., a router or an Internet service provider (ISP), does not behave as a trusted message forwarder, and fakes/alters messages sent between *Alice* and *Bob*, but still lets *Alice* and *Bob* think they are communicating to each other.

Generally speaking, communication messages in entity authentication mechanisms should be protected by certain encryption and authentication algorithms or protocols, e.g., TLS, in order to avoid leaking sensitive information to attackers for performing replay attacks and man-in-the-middle attacks.

Another method to help us defeat replay attacks is that messages communicated in an entity authentication mechanism or protocol should contain some *freshness*, e.g., a unique sequence number or an accurate time stamp. In this way, it will be detected if adversaries simply retransmit existing messages.

1.2.4 Relationships among Authentication and Encryption Algorithms

Although the authentication and encryption algorithms we have discussed in previous sections serve for their own design purposes, their applications in cloud systems may be closely related. Here we summarize the relationships among block ciphers, modes of operation, password hashing algorithms, and two-factor authentication mechanisms.

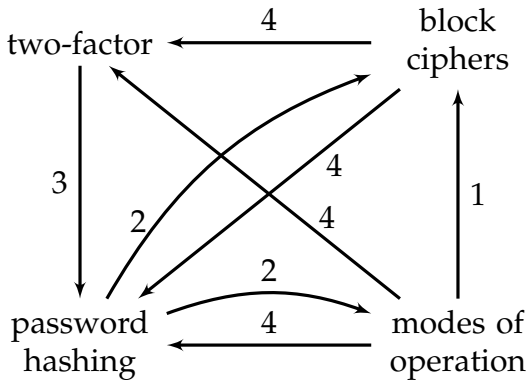

Figure 1.5: Relationships among authentication and encryption algorithms.

1. Block ciphers need to be used with modes of operation, e.g., CTR and CBC, to encrypt long messages, or with authenticated encryption modes of operation, e.g., CCM and GCM, to perform encryption and authentication simultaneously.

2. Cryptographic keys for block ciphers and modes of operation may need to be computed by using password hashing algorithms, e.g., PBKDF2 and bcrypt.

3. Two-factor authentication methods can be used to enhance the security of password-based entity authentication that relies on password hashing algorithms.

4. Block ciphers and modes of operation may be needed in entity authentication protocols in order to encrypt and authenticate passwords or two-factor information transmitted via the Internet.

Figure 1.5 illustrates the relationships described above, where arrows are numbered accordingly and each arrow denotes a "being needed by" relationship, e.g., the arrow between "block ciphers" and "modes of operation" means that modes of operation are needed when we use block ciphers.

1.3 Related Work and Our Motivations

How to protect the security of cloud systems, e.g., preventing customers' data from unauthorized access by adversaries, is one of the fundamental and important problems of cloud system designs, and it highly depends on the security levels of underlying algorithms

protecting confidentiality and authenticity. However, in practice, system designers and software developers may have restrained time and resources to learn and understand the detailed designs and principles of sophisticated cryptographic algorithms or protocols, and may make poor decisions in their system developments and put customers' data in danger. Therefore, bridging the gap between academic research and practical applications is a very important task for researchers. In this thesis, we attempt to analyze and design authentication and encryption algorithms for communication/data security and entity authentication, in order to present simple and secure choices to cloud system designers and developers.

1.3.1 Attacks on Authenticated Encryption Modes

The design of GCM is based on the CTR mode for encryption and a polynomial-based MAC scheme for authentication. Due to the important role of GCM in TLS v1.2, IEEE 802.1AE and IPsec, the security of GCM has been assessed by many researchers [51, 60, 69].

Particularly, Iwata *et al.* found a flaw in GCM's original security proofs [68], which implies the actual security bounds of GCM about authenticity and confidentiality should be much larger than the desired ones if non-96-bit nonces (IVs) are used. This drastically limits the potential application scenarios of GCM.

Moreover, the algebraic structure of GCM's underlying polynomial-based MAC scheme has been analyzed by Saarinen [107], Procter and Cid [100] recently. The polynomial-based design is also the basis for several other authentication algorithms, such as Poly1305 [20] and SGCM [107]. Procter and Cid have investigated MAC forgery attacks on these MAC schemes and revealed that *almost all* subsets of authentication keys for these polynomial-based MAC schemes are weak key classes.

Under such circumstances, further investigation on the security bounds of GCM and MAC forgeries of these polynomial-based MAC schemes would be very important for applications of GCM and future designs of authenticated encryption algorithms.

1.3.2 Meet-in-the-Middle Attacks on Block Ciphers

Meet-in-the-middle (MITM) attacks were first introduced by Diffie and Hellman for the cryptanalysis of the Data Encryption Standard (DES) [41]. The MITM technique is a generic method to analyze the high-level structures of cryptographic algorithms. The main idea of MITM attacks is that if the target algorithm can be decomposed into two small

parts and the computation of each part only involves a portion of the master key, then we can investigate the security level of each part separately and finally combine the results from both sides. Since evaluating two smaller segments usually requires much less work, the overall time complexity to analyze the complete algorithm could be decreased dramatically.

Several successful variants of MITM attacks have been developed recently, including *biclique attacks* [27, 74] that form the first single-key attacks on the full-round block ciphers AES and IDEA [78], and *splice-and-cut attacks* that construct the pre-images of MD5 [110], SHA-0 and SHA-1 [8]. In addition, the papers [36, 43] present a new idea of guessing one internal state of block ciphers and applying MITM attacks to the portions of the ciphers, although their attacks only succeed in improving the memory and data complexities, but not the time complexity, of the previous work in [65].

On the other hand, many new block cipher designs have been proposed recently, in order to provide more efficient solutions than the widely used AES without compromising overall security levels. For example, the KATAN/KTANTAN families of block ciphers [33], a lightweight stream cipher WG-7 [81], and the authenticated encryption algorithm Hummingbird-2 [50] are devised specifically for constrained environments. The block cipher PRINTcipher [75] is designed to be compact enough for integrated circuit printing. A 64-bit version block cipher, LED [59], is proposed based on the structure of AES, which has similar security evaluation but smaller implementation footprints.

Security evaluation of these new lightweight encryption algorithms, potentially by using the MITM-based approaches, is a very important task for researchers.

1.3.3 Constrained Choices of Password Hashing Designs

Although password hashing is the foundation of many real-world security systems and services, there are only limited proposals of password hashing algorithms that are well studied and widely adopted. This is due to the uncommon and demanding requirements of password hashing designs, e.g., they should be heavyweight in computation and/or memory usage [72, 98], which is an opposite goal for most cryptographic designs.

PBKDF2, a key derivation function designed and standardized by RSA Laboratories, has been the subject of extensive research and still remains the best conservative choice. PBKDF2 is a conventional design that mainly relies on iterating a pseudorandom function (usually HMAC-SHA1) a certain number of times. However, the iterative design leads to quite unaggressive usage of memory, which makes large-scale and parallel searching possible [47].

bcrypt is designed by Provos and Mazières [101], based on the block cipher Blowfish [111] with a purposefully expensive key schedule. Due to the adaptive iteration count that can be increased to make it slower, bcrypt could remain resistant to brute-force search attacks even with vast increases in computing power. Like PBKDF2, bcrypt works in-place in memory and performs poorly towards thwarting attacks using dedicated hardware.

scrypt, designed by Percival [98], is a proposal which not only offers stronger security from a theoretical point of view than the other two but also allows users to configure the amount of space in memory required to efficiently complete the algorithm. The customizable memory requirement makes it difficult for attackers to build large-scale searching circuits for scrypt, e.g., using graphics processing units (GPUs), field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs). scrypt has been selected as the underlying proof-of-work function for many cryptocurrencies, e.g., Litecoin [3] and Dogecoin [2]. However, the design of scrypt is complicated and might be error-prone to be implemented by software developers, since it involves many cryptographic primitives such as HMAC, SHA256, PBKDF2 and Salsa20/8 [22], and certain internal structures like ROMix and BlockMix.

If we could design a password hashing algorithm that is *provably secure* against common attacks, and also requires certain (adjustable) amount of memory for efficient computations, but is simpler to be implemented than the design of scrypt, it would be an attractive and useful option to be adopted by practical password-based authentication systems.

1.3.4 Quest to Enhance or Replace Passwords

With the advances of cloud services and web applications, people are likely to have more than ten accounts for social networks, email accounts, shopping websites, and various other cloud services, all with different passwords and security policies. Memorizing all passwords is both difficult and inconvenient, so people often end up with using simple passwords, or constantly forgetting the least frequently used ones.

One approach to reduce users' burden for holding multiple passwords for different cloud services is to employ an Internet-scale identity system that defines standardized mechanisms enabling the identity attributes of users to be shared among web applications and cloud services. A number of technologies and standards such as OpenID [96] and OAuth [61] have emerged to deliver an Internet-scale identity system during the past few years. But such methods like OpenID and OAuth require users to enter password and log in to a central account even if users are using a public or untrusted device, which may leak users' passwords to keyloggers or malware.

It would be very useful if we could find an innovative way of accessing cloud services, which neither involves memorizing passwords, nor adds layers of complexity for users. Bonneau *et al.* recently presented a comprehensive evaluation [30] for two decades of proposals to replace text passwords for general-purpose user authentication on the Internet. Their evaluation results have demonstrated the difficulty of replacing passwords and highlighted the research challenges towards designing a password-less authentication scheme.

1.4 Outline and Main Contributions

Chapter 2 provides a practical method, i.e., changing the counter incrementing function of the CTR mode from the modular addition to a simple operation in the finite field, in order to avoid the security proof flaw discovered by Iwata *et al.* and repair GCM. By applying this method, the security bounds of GCM can be improved by a factor of about 2^{20}. We also give security proofs for the revised mode of operation, named LGCM. Two implementation alternatives of LGCM are discussed for thwarting timing-based side-channel attacks. Preliminary results of this chapter have been published in our paper [128].

Chapter 3 analyzes MAC forgery attacks and weak key classes of polynomial-based MAC algorithms including the one used in GCM, based on the recent work of Saarinen, Procter and Cid. We reveal (and demonstrate by practical examples) that hash collisions are not necessarily required for forgeries of GCM-like polynomial-based MAC schemes, and polynomials with non-zero constant terms can be used for the attacks. These remove certain restrictions of MAC forgery attacks proposed by Procter and Cid. Based on the discoveries on MAC forgeries, we show that all non-singleton subsets (i.e., with more than one element) of authentication keys are weak key classes, if the final masking by block ciphers is computed additively. This is an extension to the previous analysis result of Procter and Cid. Furthermore, based on a special structure of GCM, we show how to turn these forgery attacks into birthday-bound-based attacks by querying the encryption oracle instead of the verification or decryption oracle. This can significantly increase success probabilities and avoid certain countermeasures. At last, we indicate that even if GCM is changed to the MAC-then-Enc paradigm to make it more difficult for adversaries to attack MAC schemes, which is one of the options mentioned in [100], our MAC forgery attacks can still work. The content of this chapter has also appeared in our paper [128].

Chapter 4 investigates a new cryptanalysis method in depth: Firstly ciphers are divided

into consecutive sub-ciphers by guessing certain intermediate states, then MITM attacks are applied to these sub-ciphers separately, and finally results are brought together to eliminate wrong secret keys. We apply this *multidimensional* approach to the KATAN block cipher family, and obtain the best cryptanalysis results so far. Our new attacks can recover the master keys of 175-round KATAN32, 130-round KATAN48, and 112-round KATAN64 faster than exhaustive search, and thus have reached many more rounds than the existing attacks. New attacks on 115-round KATAN32 and 100-round KATAN48 are also proposed in order to show that this new kind of attacks can be more efficient than the existing ones. This work has been published in our paper [126].

Chapter 5 proposes two novel password hashing algorithms, PLECO and PLECTRON, based upon several well-studied cryptographic structures and primitives. The novelty in the designs of PLECO and PLECTRON is the combination of symmetric-key and asymmetric-key algorithms that offers a twofold benefit: 1) Since the tools to cryptanalyzing asymmetric-key algorithms are quite different from those for symmetric-key ones, the composition of symmetric-key and asymmetric-key components will make cryptanalysis much harder. The basic idea is analogous to the designs of ARX (additions, rotations and XORs) based cryptographic primitives [13, 22, 93] and the block cipher IDEA [78], in which different operations are alternatively applied; 2) The asymmetric-key components not only make our scheme provably secure (the one-wayness of PLECO is as strong as the hard problem of integer factorization), but also enable server-specific computational shortcuts as a result of faster exponentiation via the Chinese Remainder Theorem when factors of moduli are known by legitimate users. In addition to describing the PLECO and PLECTRON designs in great detail, we theoretically prove their security with respect to one-wayness and collision resistance. PLECO and PLECTRON are also designed to be *sequential memory-hard* to thwart brute-force attacks using dedicated hardware. Our work on these two password hashing designs has been published in [123]. Interested readers may refer to [125] for our latest progress.

Chapter 6 presents the design of LOXIN, an innovative framework for password-less cloud authentication systems. After an initial registration process, LOXIN enables a user to access multiple cloud services or web applications with only a few taps on his/her mobile devices. This salient feature comes from the adoption of asymmetric-key, i.e., public-key, cryptosystems and cloud-based push message services. Different from most existing cloud login/authentication solutions, the servers of LOXIN are not able to generate users' credentials. Therefore, even if a LOXIN server is compromised, the

attacker cannot impersonate a user in order to access cloud services. As a potential application, we have followed the LOXIN security framework to build a password-less mobile payment solution for tackling the MintChip Challenge [89]. The main content of this chapter has published in our paper [124].

Chapter 7 summarizes the thesis and suggests potential topics for future research.

Chapter 2

Repairing the Galois/Counter Mode of Operation

This chapter investigates the provable security of the Galois/Counter Mode (GCM). As we have mentioned in Section 1.2.2, GCM has been adopted by many security standards and protocols for protecting cloud communications and storage. First, Section 2.1 describes the detailed design of GCM along with the security proof flaw discovered by Iwata *et al.* Next, a simple operation over the finite field is introduced in Section 2.2, followed by our method for repairing GCM described in Section 2.3. After that, Section 2.4 presents the methods of implementing the revised GCM for preventing side-channel attacks based on timing information. The last section summarizes the chapter.

2.1 Preliminaries

Following the paper [68], we use the following notations throughout this chapter.

- $||$ concatenates two bit-strings, e.g., $s_1||s_2$.
- $\mathsf{str}_n(x)$ denotes the n-bit binary representation of the integer x, where the leftmost bits are interpreted as the most significant bits (MSB).
- $\mathsf{int}(s)$ returns the integer converted from the bit-string s.
- 0^l denotes a bit-string consisting of l-bit 0's, where $t \geq 0$. 0^0 means an empty string. Particularly, $0^{31}1$ denotes the concatenation of 0^{31} with one 1.

- $\mathsf{len}(s)$ returns the bit-length of s.
- $\mathsf{msb}_n(s)$ represents the leftmost n bits of s.
- $\mathsf{lsb}_n(s)$ is the rightmost n bits of s.
- $|\mathcal{S}|$ denotes the cardinality of a set $\mathcal{S}$.

The function $\mathsf{inc}(s)$, where $\mathsf{len}(s) = 128$, is defined as

$$\mathsf{inc}(s) = \mathsf{msb}_{96}(s)||\mathsf{str}_{32}(\mathsf{int}(\mathsf{lsb}_{32}(s)) + 1 \bmod 2^{32}),$$

and inc^n denotes applying inc for n times.

2.1.1 Introduction to the Galois/Counter Mode

As we have mentioned in Section 1.2.2, GCM is an authenticated encryption with associated data (AEAD) scheme. GCM adopts the CTR mode for encryption, and a polynomial-based algorithm for message authentication. For simplicity, we concentrate on the version of GCM using 128-bit block ciphers, which is the major use case proposed in its specification [85]. For ciphers with other block sizes, our calculations can be easily adjusted. The finite field $\mathbb{F}_{2^{128}}$ adopted in GCM uses the generating polynomial $1 + x + x^2 + x^7 + x^{128}$.

Before describing GCM in detail, we first introduce a polynomial-based hash function $\mathsf{GHASH}_H(\cdot,\cdot)$, where H is a secret authentication key. Assume w and v are two bit-strings, where $\mathsf{len}(w) = 128(n_1 - 1) + m_1$ and $\mathsf{len}(v) = 128(n_2 - 1) + m_2$, $1 \le m_1, m_2 \le 128$, and $n_1, n_2 \ge 0$. If w is a non-empty bit-string, we segment w into $w = w_1||w_2||\cdots||w_{n_1}$, where $\mathsf{len}(w_i) = 128$ for $1 \le i \le n_1 - 1$, and $\mathsf{len}(w_{n_1}) = m_1$. If w is an empty bit-string, we define $w = w_0 = 0^0$. Similarly, if v is not 0^0, v is segmented into $v = v_1||v_2||\cdots||v_{n_2}$, where $\mathsf{len}(v_i) = 128$ for $1 \le i \le n_2 - 1$, and $\mathsf{len}(v_{n_2}) = m_2$; if v is an empty string, v is defined to be $v = v_0 = 0^0$. We also give the following notation,

$$B_i = \begin{cases} w_i & \text{for } 1 \le i \le n_1 - 1, \\ w_i||0^{128-m_1} & \text{for } i = n_1, \\ v_{i-n_1} & \text{for } n_1 + 1 \le i \le n_1 + n_2 - 1, \\ v_{i-n_1}||0^{128-m_2} & \text{for } i = n_1 + n_2, \\ \mathsf{str}_{64}(\mathsf{len}(w))||\mathsf{str}_{64}(\mathsf{len}(v)) & \text{for } i = n_1 + n_2 + 1. \end{cases}$$

The computation of $\mathsf{GHASH}_H(\cdot,\cdot)$ is defined as

$$\mathsf{GHASH}_H(w,v) = \sum_{i=1}^{n_1+n_2+1} B_i H^{n_1+n_2+2-i}, \tag{2.1}$$

where the operations in the equation (2.1) are defined over the finite field $\mathbb{F}_{2^{128}}$. Particularly, we define

$$\mathsf{GHASH}_H(s) = \mathsf{GHASH}_H(0^0, s),$$

i.e., the first parameter is an empty bit-string.

The authenticated encryption of GCM requires four bit-string inputs,

- a nonce N,
- a master key K,
- a plaintext P, and
- an associated data A,

and then produces a pair (C,T), where

- C is the ciphertext with the same length as P, and
- T is a t-bit authentication tag, where $0 < t \leq 128$.

The authenticated decryption algorithm takes N, K, A, C and T as input, and returns P if T is a valid MAC tag according to N, K, A and C, or *FAIL* if T is invalid.

The lengths of these variables should meet the following requirements [86]:

$$\begin{array}{rll} 0 \leq & \mathsf{len}(N) & \leq 2^{64}, \\ 0 \leq & \mathsf{len}(A) & \leq 2^{64}, \text{ and} \\ 0 \leq & \mathsf{len}(P) & \leq 128(2^{32}-2). \end{array}$$

We use $E_K(x)$ to denote the block cipher encryption with the master key K. Suppose $\mathsf{len}(P) = 128(n-1)+m$, where $n \geq 0$ and $1 \leq m \leq 128$. If P is a non-empty string, we segment P into a sequence of message blocks $P_1||P_2||\cdots||P_n$, where $\mathsf{len}(P_i) = 128$ for $1 \leq i \leq n-1$ and $\mathsf{len}(P_n) = m$. If P is an empty string, we define $P = P_0 = 0^0$.

Algorithm 2.1 ([86]). *The steps of the authenticated encryption process of GCM are described as follows,*

$$\begin{aligned}
H &= E_K(0^{128}),\\
N_0 &= \begin{cases} N||0^{31}1 & \textit{if } \mathsf{len}(N) = 96,\\ \mathsf{GHASH}_H(N) & \textit{if } \mathsf{len}(N) \neq 96, \end{cases}\\
N_i &= \mathsf{inc}(N_{i-1}) \textit{ for } 1 \leq i \leq n,\\
C_i &= P_i \oplus E_K(N_i) \textit{ for } 1 \leq i \leq n-1,\\
C_n &= P_n \oplus \mathsf{msb}_m(E_K(N_n))\\
C &= C_1||C_2||\cdots||C_n,
\end{aligned}$$

and the authentication tag T is computed by GMAC *defined as*

$$T = \mathsf{GMAC}_{H,t}(A, C) = \mathsf{msb}_t(\mathsf{GHASH}_H(A, C) \oplus E_K(N_0)). \tag{2.2}$$

GCM follows the Enc-then-MAC (EtM) paradigm, i.e., computing authentication tags from ciphertexts. The whole authenticated encryption process of GCM is illustrated in Figure 2.1, where the plaintext consists of two blocks, the associated data has only one block, incr is the counter incrementing function, and mult_H denotes multiplying by H in the finite field.

One important requirement when using GCM is that nonces must be distinct. Once a nonce is reused, the counter numbers N_i used in the CTR mode of encryption will be the same, and thus XORing two ciphertexts will eliminate the key stream and get information about plaintexts. Another reason of forbidding nonce reuse is explained in Joux's *forbidden attack* [69], i.e., the same nonces will result in an identical $E_K(N_0)$ used in the equation (2.2) and we can construct an equation in terms of H over the finite field, which may be easily solved, by XORing two authentication tags. However, nonces do not have to be chosen randomly. In practice, nonces can be implemented by certain simple methods, such as using incrementing counters or timestamps.

2.1.2 Attack Models and Security Definitions

For GCM with a fixed (but unknown) master key K, adversaries are given two oracles, *encryption oracle* and *decryption oracle*. Adversaries can feed (N, A, P) to the encryption oracle to get (C, T). If adversaries query the decryption oracle with (N, A, C, T), the decryption oracle will return P if T passes verification, or *FAIL* otherwise. Adversaries

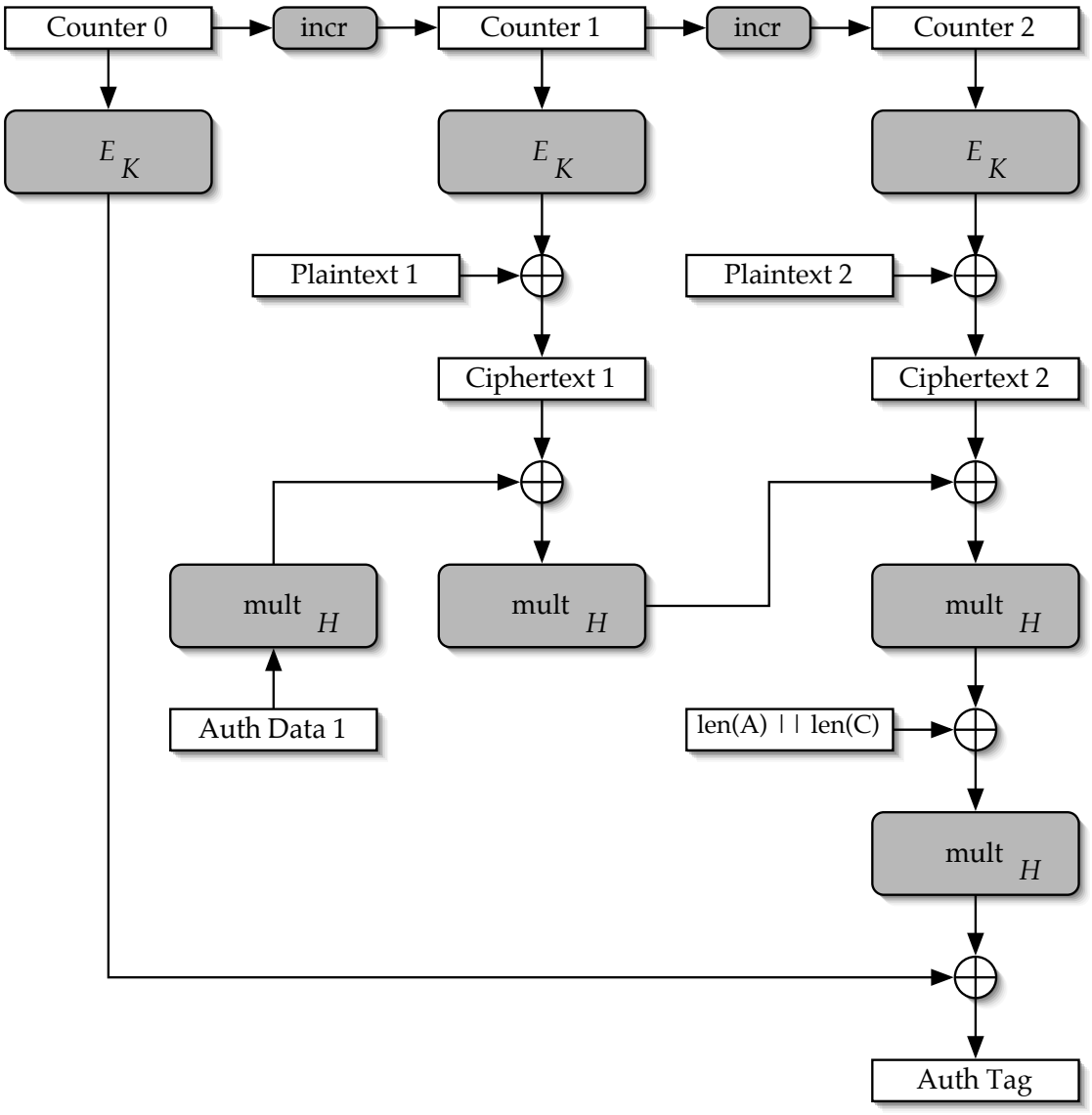

Figure 2.1: An illustration of the authenticated encryption process of GCM [85].

are assumed to be nonce-respecting, i.e., no repeating nonces are queried to the encryption oracle, which is not allowed in the GCM or CTR mode.

If adversaries target only MAC schemes, they can be given two oracles, *authentication oracle* and *verification oracle*. The authentication oracle produces T for queried (N, A, C); while the verification oracle returns *FAIL* if T is not valid for (N, A, C), or returns *PASS* otherwise.

In the setting of GCM or GMAC, if the goal of adversaries is to construct *MAC forgeries*, adversaries attempt to create a valid authentication tag T for (N, A, C), which has not been queried directly to the encryption oracle or the authentication oracle.

The security of GCM is characterized by *privacy advantage* and *authenticity advantage* of adversaries [68, 86]. Here we present the definitions of privacy and authenticity advantages given in [68]. Let $\mathcal{K}$ be the key space for a block cipher E, and GCM be written as GCM$[E, t]$ that takes E and a tag length t as parameters. For a given block cipher E, we define a GCM encryption oracle GCM-$\mathcal{E}_K$ that takes (N, A, P) and returns (C, T), and a GCM decryption oracle GCM-$\mathcal{D}_K$ that takes (N, A, C) and returns P or *FAIL*, where K is the secret key in use. We also define a random-bit oracle \$ that takes (N, A, P) and returns random bits $(C, T) \xleftarrow{\$} \{0,1\}^{\mathsf{len}(P)+t}$, where t is the tag length.

Definition 2.1 (Privacy Advantage [68])**.** *Assuming an adversary $\mathcal{A}$ is nonce-respecting, we define the privacy advantage of $\mathcal{A}$ attacking GCM as*

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{GCM}[E,t]}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\text{GCM-}\mathcal{E}_K} \Rightarrow 1] - \Pr[\mathcal{A}^{\$} \Rightarrow 1].$$

Definition 2.2 (Authenticity Advantage [68])**.** *Assuming an adversary $\mathcal{A}$ is nonce-respecting when querying the encryption oracle* GCM-$\mathcal{E}_K$*, we define the authenticity advantage of $\mathcal{A}$ attacking GCM as*

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{GCM}[E,t]}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\text{GCM-}\mathcal{E}_K,\text{GCM-}\mathcal{D}_K} \textit{ forges}].$$

2.1.3 A Flaw in the Security Proofs of the Galois/Counter Mode

Iwata *et al.* have discovered a flaw [68] in the security proofs of GCM given by McGrew and Viega in [86], where the core issue originates from the assumption that the following equation

$$\mathsf{inc}^{r_1}(\mathsf{GHASH}_H(N^{(a)})) = \mathsf{inc}^{r_2}(\mathsf{GHASH}_H(N^{(b)})) \tag{2.3}$$

has at most $l_N + 1$ solutions for any given r_1, r_2, and two nonces $N^{(a)}$ and $N^{(b)}$, where $0 \leq r_1, r_2 \leq 2^{32} - 2$, $N^{(a)} \neq N^{(b)}$, and l_N is the maximum number of blocks for nonces, i.e., $l_N + 1$ is the maximum degree of $\mathsf{GHASH}_H(N^{(a)})$ and $\mathsf{GHASH}_H(N^{(b)})$ in terms of H.

However, Iwata *et al.* discovered that inc may be translated to multiple distinct forms in the finite field, such that the equation (2.3) may have many more solutions than the desired $l_N + 1$, as described in the following result.

Result 2.1 ([68]). *For a randomly chosen H, the probability for the equation (2.3) to hold is at most*

$$\frac{2^{22}(l_N + 1)}{2^{128}}.$$

Furthermore, for n queries to the encryption oracle with the nonces $N^{(i)}$'s, where $1 \leq i \leq n$, the probability of having a collision on counter numbers, i.e., $N_{r_1}^{(a)} = N_{r_2}^{(b)}$ for certain r_1, r_2, a and b, is at most

$$\frac{2^{22}(n-1)(\sigma + n)(l_N + 1)}{2^{128}}, \tag{2.4}$$

where $0 \leq r_1, r_2 \leq 2^{32} - 2$, $1 \leq a, b \leq n$, the total length of plaintexts is at most σ blocks, l_N is the maximum number of blocks for nonces, and $N^{(a)}$ and $N^{(b)}$ are the corresponding nonces for the counter numbers $N_{r_1}^{(a)}$ and $N_{r_2}^{(b)}$ respectively.

Iwata *et al.* have also given new security bounds for privacy and authenticity advantages of GCM as follows, where the block cipher E is treated as a pseudorandom permutation PRP that takes 128-bit blocks as input and also generates 128 bits as output.

Result 2.2 ([68]). *For any adversary $\mathcal{A}$ that makes at most q encryption queries and q' decryption queries, where the total length of plaintexts is at most σ blocks, and l_N, l_A are the maximum numbers of blocks for nonces and inputs, respectively, we have*

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{GCM}[\mathsf{PRP},t]}(\mathcal{A}) \leq \frac{0.5(\sigma + q + 1)^2}{2^{128}} + \frac{2^{22} q(\sigma + q)(l_N + 1)}{2^{128}}, \tag{2.5}$$

and

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{GCM}[\mathsf{PRP},t]}(\mathcal{A}) \leq \frac{0.5(\sigma + q + q' + 1)^2}{2^{128}} + \frac{2^{22}(q + q' + 1)(\sigma + q)(l_N + 1)}{2^{128}} + \frac{q'(l_A + 1)}{2^t}. \tag{2.6}$$

The values of the equations (2.5) and (2.6) are generally dominated by their second terms, since they have a large constant 2^{22} that does not exist in GCM's original (flawed) security proofs.

2.2 A Simple Operation over the Finite Field

Since the flaw of GCM's security proofs originates from the operation inc as explained in the previous section, we aim to replace the functionality of inc in the GCM mode with another operation in the finite field.

Consider $w \cdot x$, where w is a primitive element of $\mathbb{F}_{2^n}$. It is clear that the outputs of $w \cdot x$ consist of two cycles, namely (0) and $(1, w, \ldots, w^{2^n-2})$. To combine these two cycles into one, we define a new function L_w as

$$L_w(x) = \begin{cases} w \cdot x & \text{if } x = w^i,\ 0 \le i \le 2^n - 3, \\ 0 & \text{if } x = w^{2^n-2}, \\ 1 & \text{if } x = 0. \end{cases} \tag{2.7}$$

The following theorem is important for our discussions in this chapter.

Theorem 2.1. *Let $\mathbb{F}_{2^n}$ be a finite field, w be a primitive element of $\mathbb{F}_{2^n}$, r_{max} be an integer, where $0 \le r_{max} \le 2^n - 1$, and $L_w(x)$ be a mapping over $\mathbb{F}_{2^n}$ defined as the function (2.7). Given any two mappings $f(x), g(x)$ over $\mathbb{F}_{2^n}$ that satisfy*

- *$f(0) = 0$ and $g(0) = 0$, and*
- *$w^s f(x) + g(x)$ is not the zero polynomial for any s, where $0 \le s \le r_{max}$,*

then we have

$$\max_{0 \le r \le r_{max}} |\{x \in \mathbb{F}_{2^n} \mid L_w^r(f(x)) + g(x) = 0\}| \le 4d,$$

where $d = \max\{\deg(f), \deg(g)\}$.

Proof. Now we consider the number of solutions of the equation

$$L_w^r(f(x)) + g(x) = 0 \tag{2.8}$$

for a given r, where $0 \le r \le r_{max} \le 2^n - 1$ and $f(x), g(x)$ satisfy the properties given in the theorem. We divide the solutions into several cases below. For an element $y \in \mathbb{F}_{2^n}$:

(i) If $f(y) = 0$ and $L_w^r(f(y)) = 0$, we must have that $g(y) = 0$ such that (2.8) holds. We denote the set of such y by Ψ_1;

(ii) If $f(y) = 0$ and $L_w^r(f(y)) \neq 0$, then it is not difficult to check that $L_w^r(f(y)) = w^{r-1}$, and thus $g(y) = w^{r-1}$ such that (2.8) holds. Let us denote the set of such y by Ψ_2;

(iii) If $f(y) \neq 0$ and $L_w^r(f(y)) = 0$, then from (2.8) we must have $g(y) = 0$. These elements y are denoted by Ψ_3;

(iv) If $f(y) \neq 0$ and $L_w^r(f(y)) \neq 0$, assume $f(y) = w^{r_1}$ and $L_w^r(f(y)) = w^{r_2}$, where $0 \leq r_1, r_2 \leq 2^n - 2$, and then:

(iv.a) If $r_1 \leq r_2$, then (2.8) leads to $g(y) = L_w^r(f(y)) = w^r f(y)$. Denote such elements by a subset Ψ_4;

(iv.b) If $r_1 > r_2$, similarly we have $g(y) = w^{r-1} f(y)$ and denote such elements by Ψ_5.

There is no difficulty to see that the solution set Ψ of (2.8) is the union of Ψ_i for $1 \leq i \leq 5$; and Ψ_i's are pairwise disjoint. Therefore, we have that

$$|\Psi| = |\Psi_1| + |\Psi_2| + \cdots + |\Psi_5|.$$

Firstly, assume $g(x)$ is not the zero polynomial. It is easy to see that both Ψ_1 and Ψ_3 are disjoint subsets of $G = \{x \in \mathbb{F}_{2^n} \mid g(x) = 0\}$, so we have

$$|\Psi| \leq |\Psi_2| + |\Psi_4| + |\Psi_5| + |G|.$$

Let us consider the size of each set individually. Here we utilize the simple fact that the size of a solution set is at most the degree of one of its defining polynomials, if the polynomial is not the zero polynomial.

- $g(x) + w^{r-1}$ is not the zero polynomial since the constant term of $g(x)$ is zero. Therefore, $|\Psi_2|$ is safely less than or equal to the degree of the equation $g(x) = w^{r-1}$.
- $|\Psi_4|$ is less than or equal to the degree of $g(x) = w^r f(x)$, since $g(x) + w^r f(x)$ is apparently not the zero polynomial due to the properties defined in the theorem.
- $|\Psi_5|$ is less than or equal to the degree of $g(x) = w^{r-1} f(x)$. Similar as $w^r f(x) + g(x)$, $w^{r-1} f(x) + g(x)$ is not the zero polynomial.
- $|G|$ is less than or equal to the degree of $g(x) = 0$.

Therefore, $|\Psi| = |\Psi_2| + |\Psi_4| + |\Psi_5| + |G| \leq 4d$.

Secondly, if $g(x)$ is the zero polynomial, then $f(x)$ will not also be the zero polynomial since under such circumstance $w^s f(x) + g(x)$ would always be zero for any s. It is clear that $\Phi_i = \emptyset$ for $i = 2, 4, 5$ if $g(x) = 0$, and then in this case

$$|\Psi| = |\Psi_1| + |\Psi_3|.$$

Similarly as above, we consider each individual set.

- $|\Psi_1|$ is less than or equal to the degree of $f(x) = 0$.
- If $f(y) \neq 0$ and $L_w^r(f(y)) = 0$, then $f(y) = w^{2^n-2}$ due to the definition of $L_w(x)$. Because $f(x)$'s constant term is zero, $f(x) + w^{2^n-2}$ must not be the zero polynomial, and then $|\Psi_3|$ is less than or equal to the degree of $f(x) = w^{2^n-2}$.

In sum, in this case, $|\Psi| = |\Psi_1| + |\Psi_3| \leq 2d$.

We complete the proof. □

2.3 Repairing the Galois/Counter Mode and Its Security Bounds

Larger security bounds for the advantages of adversaries than the original ones imply that adversaries will have higher bounds on probabilities of recovering plaintexts or constructing MAC forgeries. Therefore, it will be useful if we can repair the design of GCM and tighten its security bounds. Here we propose a revision of GCM such that the large constant 2^{22} in the equations (2.5) and (2.6) can be reduced to 2^2.

It is known that the detailed design of the counter incrementing function of the CTR mode is not important as long as counter numbers are produced uniquely [84]. If the underlying block cipher is ideal, i.e., treated as a pseudorandom permutation PRP, $\mathsf{PRP}(L_w^r(s))$ is indistinguishable from $\mathsf{PRP}(\mathsf{inc}^r(s))$. Therefore, the CTR mode encryption in GCM, without considering the initial counter generation methods, will have the same security properties as GCM's original design if inc is replaced by L_w.

We propose the following revised design of GCM.

Algorithm 2.2. *The steps of the authenticated encryption process of the revised GCM, denoted by LGCM, are as follows,*

$$\begin{aligned} H &= E_K(0^{128}), \\ N_0 &= \mathsf{GHASH}_H(N), \\ N_i &= L_w^i(N_0) \text{ for } 1 \le i \le n, \\ C_i &= P_i \oplus E_K(N_i) \text{ for } 1 \le i \le n-1, \\ C_n &= P_n \oplus \mathsf{msb}_m(E_K(N_n)), \\ C &= C_1||C_2||\cdots||C_n, \\ T &= \mathsf{GMAC}_{H,t}(A, C), \end{aligned}$$

where the notations are the same as the ones used in Algorithm 2.1, and

$$\mathsf{len}(N) = const \tag{2.9}$$

for a predefined non-negative integer, const.

Please note that nonces are always processed by GHASH regardless of nonces' lengths, for simplicity of security proofs.

Based on Theorem 2.1, we can have the following lemma.

Lemma 2.1. *Randomly choosing an authentication key H, the probability of having*

$$L_w^{r_1}(\mathsf{GHASH}_H(N_1)) = L_w^{r_2}(\mathsf{GHASH}_H(N_2)) \tag{2.10}$$

is no more than $4(l_N+1)/2^{128}$ for any given r_1, r_2, N_1 and N_2, where $0 \le r_1, r_2 \le 2^{32}-2$, l_N is the maximum number of blocks for nonces, $N_1 \ne N_2$, and $\mathsf{len}(N_1) = \mathsf{len}(N_2)$.

Proof. Since $N_1 \ne N_2$ and $\mathsf{len}(N_1) = \mathsf{len}(N_2)$, the nonces N_1 and N_2 cannot both be the empty string, which implies $\mathsf{len}(N_1) = \mathsf{len}(N_2) > 0$.

Without loss of generality, we assume $r_2 \le r_1$. The equation (2.10) is equivalent to

$$L_w^{r_1-r_2}(\mathsf{GHASH}_H(N_1)) = \mathsf{GHASH}_H(N_2). \tag{2.11}$$

The maximum degree of the polynomials $\mathsf{GHASH}_H(N_1)$ and $\mathsf{GHASH}_H(N_2)$ in terms of H is $l_N + 1$.

Recall that $\mathsf{GHASH}_H(N)$ is a polynomial of H with zero constant term. We denote such a polynomial as $G_N(x) = \mathsf{GHASH}_x(N)$ for a given N, and thus $G_N(0) = \mathsf{GHASH}_0(N) = 0$ for any N. Therefore, $G_{N_1}(0) = 0$ and $G_{N_2}(0) = 0$.

Let w be a primitive element in the finite field. $w^s G_{N_1}(x) + G_{N_2}(x) = w^s \mathsf{GHASH}_x(N_1) + \mathsf{GHASH}_x(N_2)$ is not the zero polynomial, for any given s where $0 \le s \le 2^{32}-2$, because:

(i) If $s = 0$, $G_{N_1}(x) + G_{N_2}(x)$ is apparently not the zero polynomial, since $N_1 \neq N_2$.

(ii) Recall that the coefficient of the term x in the polynomial $G_N(x) = \mathsf{GHASH}_x(N)$ is $\mathsf{int}(0^{64}||\mathsf{str}_{64}(\mathsf{len}(N))) = \mathsf{len}(N)$. If $1 \leq s \leq 2^{32} - 2$, $w^s G_{N_1}(x) + G_{N_2}(x)$ will not become the zero polynomial, since $w^s\mathsf{len}(N_1) \neq \mathsf{len}(N_1) = \mathsf{len}(N_2)$, and XORing $w^s\mathsf{len}(N_1)$ and $\mathsf{len}(N_2)$ forms the coefficient of the term x in $w^s G_{N_1}(x) + G_{N_2}(x)$.

By applying Theorem 2.1, we know the probability for the equation (2.11) to hold is at most $4(l_N + 1)/2^{128}$ for a randomly chosen H. □

We define LGCM$[E, t]$ similarly as GCM$[E, t]$ in Section 2.1.2. LGCM$[E, t]$ takes E and a tag length t as parameters. Now we can give the security bounds of LGCM as follows.

Theorem 2.2. *For any adversary $\mathcal{A}$ that makes at most q encryption queries and q' decryption queries, where the total length of plaintexts is at most σ blocks, and l_N, l_A are the maximum numbers of blocks for nonces and inputs, respectively, we have*

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathrm{LGCM}[\mathsf{PRP},t]}(\mathcal{A}) \leq \frac{0.5(\sigma + q + 1)^2}{2^{128}} + \frac{4q(\sigma + q)(l_N + 1)}{2^{128}}, \tag{2.12}$$

and

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{LGCM}[\mathsf{PRP},t]}(\mathcal{A}) \leq \frac{0.5(\sigma + q + q' + 1)^2}{2^{128}} + \frac{4(q + q' + 1)(\sigma + q)(l_N + 1)}{2^{128}} + \frac{q'(l_A + 1)}{2^t}. \tag{2.13}$$

Proof. The proofs of Theorems 1 and 2 in [68] can be carried over, if we use Lemma 2.1 in this chapter to replace the original probability statement of counter number collisions. □

Some remarks on the equation (2.9) in Algorithm 2.2 are given as follows. In order to obtain the above results of Lemma 2.1 and Theorem 2.2, the equation (2.9) is actually not a necessary condition. We only need to restrict the choices of nonces such that $w^s f(x) + g(x)$ does not become a zero polynomial. For example, a practical application may require the value of a nonce block, e.g., the rightmost or last block, to be a fixed non-zero value, so $w^s f(x)$ will not match other valid nonces. The other, slightly flexible, method is that the choices of nonces must satisfy both: 1) By writing $\mathsf{len}(N) = 128i + j$, where i, j are non-negative integers and $j < 128$, j is unique for every possible i; 2) The left most block of each nonce is non-zero. There might be many other less restricted conditions than the equation (2.9), but (2.9) should be one of the simplest to be implemented in practice.

We also want to make a note here that it might be possible to directly adopt $w \cdot x$ instead of $L_w(x)$ to generate counter numbers since the probability for GHASH to output zero is low, but the security proofs for GCM may require to be largely rewritten and new bounds might have different formats as the existing ones.

2.4 Implementations against Timing-Based Side-Channel Attacks

The function (2.7) has vulnerabilities for timing-based side-channel attacks since its computations will have inconsistent times for different inputs. To minimize such effects, we may use the following equations to replace (2.7) in practical implementations.

$$\begin{aligned} y &= w \cdot x, \\ L_w(x) &= \begin{cases} 1 & \text{if } y = 0, \\ 0 & \text{if } y = 1, \\ y & \text{otherwise.} \end{cases} \end{aligned} \tag{2.14}$$

Alternatively, $L_w(x)$ can be written as

$$\begin{aligned} L_w(x) &= w \cdot x + y(x), \text{ where} \\ y(x) &= \begin{cases} 1 & \text{if } x = 0 \text{ or } x = w^{2^n-2}, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{2.15}$$

The equations (2.14) and (2.15) would have closer computational time costs for different branches than the equation (2.7).

2.5 Summary

In this chapter, we have revisited the provable security of GCM, and provided a practical method to fix GCM with non-96-bit nonces, in order to avoid the flaw of security proofs discovered by Iwata *et al.* This method can improve the overall security bounds of GCM by a factor of about 2^{20}. We have also presented security proofs for the revised mode of operation, namely LGCM, and alternative implementations of LGCM that are useful for thwarting timing-based side-channel attacks.

Due to the important role of GCM in various encryption and authentication protocols, such as TLS v1.2, IEEE 802.1AE and IPsec, we recommend that GCM is only used with 96-bit nonces. For instance, a variant of GCM has been introduced by Aoki and Yasuda in [9], which only accepts a fixed-length nonce. Our revised design LGCM is recommended if certain application scenarios prefer using non-96-bit nonces. In the current standard about GCM used in TLS [34], a portion of the nonce is generated by applying HMAC to certain handshake information. It is applicable to choose session-specific non-repeating data in TLS, including the handshake information, as the nonce used for LGCM.

Chapter 3

Forgery Attacks and Weak Keys of Polynomial-Based MAC Algorithms

This chapter analyzes polynomial-based MAC algorithms. As we have described in Section 2.1.1, the MAC algorithm used in GCM is a polynomial-based design. Firstly, Section 3.1 gives the background knowledge about several existing attacks on polynomial-based MAC algorithms. Secondly, Section 3.2 presents our improved forgery attacks on polynomial-based MAC schemes, and Section 3.3 studies the weak key classes of GCM-like MAC schemes. Section 3.4 shows how to turn these forgery attacks on GCM into birthday attacks to improve their success probabilities. The attacks on a revised version of GCM in the MAC-then-Enc paradigm are discussed in Section 3.5. Finally, the last section summarizes this chapter.

3.1 Preliminaries

This section gives the background knowledge about polynomial-based MAC algorithms and describes several existing attacks on them. Notations in this chapter follow the ones defined in Section 2.1.

For simplicity, in the context of GCM, the associated data A, the plaintext P and the ciphertext C are considered to be multiples of 128 bits, and the nonce N to be a multiple of 128 bits if N is not 96-bit, such that all inputs do not need to be padded. If not stated explicitly, A is regarded as an empty bit-string.

Furthermore, following the notation in [100], the indices of input blocks are reversed, e.g., $P = P_n||P_{n-1}||\cdots||P_1$ instead of $P = P_1||P_2||\cdots||P_n$, for convenience of polynomial representations.

3.1.1 Polynomial-Based MAC Algorithms

As mentioned in the previous chapter, the Galois/Counter Mode (GCM) is an authenticated encryption with associated data (AEAD) mode, which is adopted in many important cryptographic schemes and protocols. The designs of GMAC and GHASH in GCM are based on the *evaluation hash* [115].

Let $\mathbb{F}$ be a finite field of characteristic 2, $H \in \mathbb{F}$ be the authentication key, and $M = M_m||M_{m-1}||\cdots||M_1$ be a message to be authenticated, where $M_i \in \mathbb{F}$. Define a polynomial $g_M(x) \in \mathbb{F}[x]$ as

$$g_M(x) = \sum_{i=1}^{m} M_i x^i.$$

Then the function $h_H(M) = g_M(H)$ is called the *evaluation hash*. The hash function outputs are masked by block cipher encryptions to produce the authentication tags, in the ways such as $E_K(N) \oplus h_H(M)$ and $E_K(h_H(M))$. Poly1305 [20], and the MAC schemes in GCM and SGCM [107] are all within this framework.

3.1.2 Existing Attacks on Polynomial-Based MAC Algorithms

Procter and Cid have studied the weak key classes and forgery attacks of polynomial-based MAC schemes, including the one used in GCM [100]. They have provided a more general model upon Saarinen's cycling attack [107]. We summarize the main observation by Procter and Cid in [100] as follows. We include a short proof of their result, as it is the fundamental for our discussions in the subsequent sections.

Result 3.1 ([100])**.** *With the same notations as the ones in Section 3.1.1, if there exists a polynomial $f(x) \in \mathbb{F}[x]$ without a constant term, such that $f(H) = 0$, then forgeries of MAC schemes based on the evaluation hash $h_H(x)$ can be made by using $f(x)$.*

Proof. Assume

$$f(x) = \sum_{i=1}^{n} F_i x^i,$$

and $F = F_n||F_{n-1}||\cdots||F_1$. Given a message M, we have

$$h_H(M \oplus F) = g_{M\oplus F}(H) = g_M(H) \oplus f(H) = g_M(H) = h_H(M),$$

where the shorter one of M and F in $M \oplus F$ is left-padded with zeros. We obtain a collision on the evaluation hash, and thus a forgery of the MAC scheme. □

If GCM is the attack target, after obtaining a valid tuple (N, C, T) by eavesdropping or active querying, adversaries query the verification oracle about $(N, C \oplus F, T)$. If the result is not *FAIL*, then a valid MAC is forged. Please note that the polynomial $f(x)$ always has x as its factor, and is in the ideal $\langle x^2 \oplus Hx\rangle$ of the polynomial ring $\mathbb{F}[x]$.

For an unknown H, the success probability of MAC forgeries is directly related to the choice of $f(x)$. Procter and Cid have proposed three ways to select $f(x)$: (1) The first way is to use $f(x) = x\prod_i(x \oplus H_i)$ to involve as many H_i as desired; (2) The second way is based on irreducible factors of $x^{2^{128}} \oplus x$, which includes Saarinen's cycling attack as a special case; (3) The third is just using random polynomials.

Moreover, based on these analyses, Procter and Cid point out that almost any subset of the authentication key space of these polynomial-based MAC schemes is a weak key class.

Analysis of a cryptographic algorithm's *weak keys* is a very important assessment. Handschuh and Preneel have given a theoretical definition of weak keys for symmetric-key cryptosystems in [60]: "A class of keys is called *weak* if for members of the class the algorithm behaves in an unexpected way and if it is easy to detect whether a particular key belongs to this class." For example, for a MAC algorithm, the unexpected behavior may be that MAC forgeries can be made in a very high probability. Moreover, to determine whether a key is in the class $\mathcal{K}$, the number of queries has to be fewer than the exhaustive search's, i.e., $|\mathcal{K}|$.

Result 3.2 ([100])**.** *Let $\mathcal{H}$ be a subset of the authentication key space of the MAC scheme based on the evaluation hash. If $|\mathcal{H}| \geq 2$ and $0 \in \mathcal{H}$, or $|\mathcal{H}| \geq 3$, then $\mathcal{H}$ is weak.*

Proof. If $|\mathcal{H}| \geq 2$ and $0 \in \mathcal{H}$, one query forged by $f(x) = x\prod_i(x \oplus H_i)$ can be fed into the verification oracle, where $H_i \in \mathcal{H}$. To further determine whether 0 is in the set $\mathcal{H}$, two queries by distinct $f(x) \in \langle x^2 \oplus Hx\rangle$ have to be made, so all elements in a subset $|\mathcal{H}| \geq 3$ can be detected by using two queries. □

3.2 New Forgery Attacks on Polynomial-Based MAC Algorithms

The MAC forgery attacks proposed by Procter and Cid are constructed upon hash collisions, and one of the attacks' restrictions is that the chosen polynomial $f(x)$ should always have x as a factor, or equivalently do not have a constant term. We demonstrate below how to create MAC forgeries not based on hash collisions, and without the zero constant term restriction.

For the MAC schemes as in GCM and SGCM, whose final masking by block ciphers is computed additively, we give the following theorem, where the notations are the same as above.

Theorem 3.1. *Given any polynomial $q(x) \in \mathbb{F}[x]$ such that $q(H) = 0$, for the evaluation hash based MAC scheme $T = h_H(M) \oplus E_K(N)$, a MAC forgery can be constructed by using $q(x)$.*

Proof. Let $q(x) = q^*(x) \oplus Q_0$, where the Q_0 is the constant term, and Q^* be the concatenation of other coefficients as $Q_n||Q_{n-1}|| \cdots ||Q_1$. Since $q(H) = 0$, we have

$$T = h_H(M) \oplus E_k(N) = h_H(M) \oplus E_k(N) \oplus q(H),$$

which implies

$$\begin{aligned} T \oplus Q_0 = &\ h_H(M) \oplus q^*(H) \oplus E_k(N) \\ = &\ g_M(H) \oplus q^*(H) \oplus E_k(N) \\ = &\ g_{M \oplus Q^*}(H) \oplus E_k(N). \end{aligned}$$

This means if we know a polynomial $q(x)$ such that $q(H) = 0$, we can XOR coefficients of $q(x)$'s non-constant terms with the captured message, to obtain a valid tuple as $(N, M \oplus Q^*, T \oplus Q_0)$, if the authentication tag T is computed as $h_H(M) \oplus E_k(N)$. □

Please note that the method in the above proof does not rely on a hash collision, and the constant term Q_0 is not required to be zero. We also want to mention that Theorem 3.1 leads us to an extension to the original analysis of Procter and Cid on weak key classes, which is discussed in the next section.

A practical attack example on GCM with AES-128 and 128-bit authentication tags, by using the method in Theorem 3.1 (along with a length extension technique), is given as follows. The associated data A is always considered as empty. We use the same representations as the test vectors in GCM's specification [85], e.g., 1 in $\mathbb{F}_{2^{128}}$ is represented as

80000000000000000000000000000000,

and longer strings will written in multiple lines.

We take the following values for the authenticated encryption of GCM. The lengths of P and C are 128 bits, i.e., one block.

K	71eebc49c8fb773b2224eaff3ad68714
N	07e961e67784011f72faafd95b0eb640
	89c8de15ad685ec57e63d56e679d3e20
	2b18b75fcbbec3185ffc41653bc2ac4a
	e6ae8be8c85636f353a9d19a86100d0b
P	705da82292143d2c949dc4ba014f6396
H	d27430c121f14d4ddfecb38acaffec53
C	251ccc6d2c45540cac4fde8b1e36802d
T	be2da05993fbde00421c1d8eaaaea373

Suppose we have a subset of authentication keys $\mathcal{H} = \{H_1, H_2, H_3\}$, whose values are as follows.

H_1	d27430c121f14d4ddfecb38acaffec53
H_2	00000000000000000000000000000001
H_3	00000000000000000000000000000002

By constructing the polynomial

$$q(x) = \sum_{i=0}^{3} Q_i x^i = \prod_{i=1}^{3} (x \oplus H_i),$$

we can get the values for Q_i's.

Q_3	80000000000000000000000000000000
Q_2	d27430c121f14d4ddfecb38acaffec50
Q_1	c488aa211ab5dccec9c440bc33fc47b3
Q_0	5bb5716dc4b4687a06f15f10d62613ee

Please note $q(x)$ is a polynomial with a non-zero constant term, i.e., $Q_0 \neq 0$.

Then compute $\alpha = (1 \oplus 2)/Q_0 =$ 7ef05dd871ead7e7f8e79d7d9343a170, such that $\alpha \cdot Q_1 \oplus 1$ will match the length of new message, i.e., 2. Construct the new ciphertext $C' = (\alpha \cdot Q_3)||(C \oplus \alpha \cdot Q_2)$, and the authentication tag $T' = T \oplus \alpha \cdot Q_0$.

C'	7ef05dd871ead7e7f8e79d7d9343a170
	7ccbd8dbfca54d785f5662d48c7eef81
T'	8b53b318750a2e948459b204e47629b4

(N, C', T') passes the verification, and thus we complete a MAC forgery with length extension by using a polynomial with a non-zero constant term.

For the sake of completeness, we also give the following theorem, which works for both $E_K(N) \oplus h_H(M)$ and $E_K(h_H(M))$.

Theorem 3.2. *Given any polynomial $q(x) \in \mathbb{F}[x]$ such that $q(H) = 0$, a forgery can be made on the MAC schemes based on the evaluation hash, by using $\alpha(x)q(x)$, where $\alpha(x)$ is a polynomial without a constant term.*

Proof. Since $q(H) = 0$, we have $\alpha(H)q(H) = 0$. Because $\alpha(0) = 0$, $\alpha(0)q(0) = 0$. Therefore, we can apply the same method in Result 3.1 to construct hash collisions and thus MAC forgeries. □

Theorem 3.2 can be seen as covered by the analysis of Procter and Cid, since $\alpha(x)q(x)$ is still in the ideal $\langle x^2 \oplus Hx \rangle$. However, Theorem 3.2 is insufficient to deduce Theorem 3.3, which is supported by Theorem 3.1, in the next section about weak key classes.

3.3 All Non-singleton Subsets of Keys are Weak

To detect whether an authentication key H is in a subset $\mathcal{H}$ of the key space, the number of queries should be less than $|\mathcal{H}|$. If $|\mathcal{H}| = 2$, only one query can be made, and thus whether the key in use is zero cannot be determined by using polynomials in $\langle x^2 \oplus Hx \rangle$, since it will need at least two queries. However, based on the analysis of Theorem 3.1, we may use polynomials in $\langle x \oplus H \rangle$ instead of $\langle x^2 \oplus Hx \rangle$ to make one query and determine whether the authentication key is in $\mathcal{H}$.

Theorem 3.3. *For an evaluation hash based MAC scheme, $T = E_K(N) \oplus h_H(M)$, if given a valid tuple (N, M, T), then making one query to the verification oracle is enough to determine whether the authentication key $H \in \mathbb{F}$ in use is in a subset of keys $\mathcal{H} = \{H_1, H_2, \cdots, H_n\} \subseteq \mathbb{F}$.*

Proof. First define a polynomial

$$q(x) = \sum_{i=0}^{n} Q_i x^i = \prod_{i=1}^{n} (x \oplus H_i),$$

where $Q_i \in \mathbb{F}$ for $0 \leq i \leq n$. Let $M' = M \oplus Q^*$ and $T' = T \oplus Q_0$ with zero left-padding for shorter strings, where $Q^* = Q_n||Q_{n-1}|| \cdots ||Q_1$. Query the verification oracle with the tuple (N, M', T'). If the verification oracle does not return *FAIL*, the authentication key H in use is known to be in $\mathcal{H}$. H is not in $\mathcal{H}$ if *FAIL* is returned.

It is easy to see H is in $\mathcal{H}$ if and only if (N, M', T') passes. If H is in $\mathcal{H}$, then $q(H) = 0$, and thus (N, M', T') is valid. On the other hand, the validity of (N, M', T') implies $q(H) = 0$, so H must be a root of $q(x) = 0$, which is among all the elements of $\mathcal{H}$. □

The steps in Theorem 3.3 are similar to those in [100], except the absence of the steps to determine whether 0 is in $\mathcal{H}$.

Based on Theorem 3.3, we have the following corollary about weak key classes.

Corollary 3.1. *For an evaluation hash based MAC scheme as $T = E_K(N) \oplus h_H(M)$, any subset, $\mathcal{H}$, of its authentication key space is weak if $|\mathcal{H}| \geq 2$.*

Proof. Due to Theorem 3.3, after obtaining a valid tuple (N, M, T) by passive eavesdropping, whether the authentication key H in use is in the subset $\mathcal{H}$ can be determined by only one query, which is efficient compared with the size of the subset, i.e., $1 < |\mathcal{H}|$.

Moreover, once H is known to be in the subset $\mathcal{H}$, H is a solution for $q(x) = \prod_{i=1}^{n}(x \oplus H_i) = 0$, where H_i's are all elements of $\mathcal{H}$. Then the polynomial $\alpha(x)q(x)$ with an arbitrary non-zero $\alpha(x)$ can be used to construct more MAC forgeries. □

3.4 New Birthday-Bound-Based MAC Forgery Attacks on GCM

The original forgery attacks on polynomial-based MAC schemes, including our attacks described in Section 3.2, target algebraic properties of underlying evaluation hash functions, e.g., GHASH in the case of GCM. The forged queries cannot be fed to the encryption oracle directly because two queries with identical nonces are forbidden.

The work by Iwata *et al.* (as mentioned in Section 2.1.3) reminds us that GCM has a very special design, in which GHASH is reused for generating initial counter numbers if $\mathsf{len}(N) \neq 96$. This makes GHASH attackable in the encryption oracle. Precisely, assuming $H \neq 0$, the attack consists of the following three steps:

1. Either passively or actively obtain a valid tuple (N, P, C), where $\mathsf{len}(N) \neq 96$. Please note that we do not need the authentication tag T here.

2. Construct a polynomial $q(x)$, and properly apply $x^d q(x)$ to N to derive N', where $d \geq 1$. Feed the pair (N', P) to the encryption oracle, and get the corresponding ciphertext C'. If $C' = C$, we know that $q(H) = 0$.

3. Apply $q(x)$ to other captured messages and tags to construct more forgeries, or recover the authentication key by binary search or solving $q(x) = 0$.

If $H = 0$, the outputs of GHASH will always be zero, and thus it can be easily detected.

One advantage of targeting the encryption oracle is that we can collect all query results into a set to perform birthday attacks. For any query to the encryption oracle, we can always get its corresponding ciphertext and tag as long as the nonce is not previously queried.

Following the notations in Algorithm 2.1, we collect $E_K(N_1)$'s, which are derived by XORing P_1's with C_1's, into a set $\mathcal{S}$. If a collision occurs in $\mathcal{S}$, e.g., $E_K(N_1^{(a)}) = E_K(N_1^{(b)})$, where $N_1^{(a)}$ and $N_1^{(b)}$ are the corresponding first counter numbers for the nonces $N^{(a)}$ and $N^{(b)}$, then we know $N_1^{(a)} = N_1^{(b)}$ as well. Hence a collision $\mathsf{GHASH}_H(N^{(a)}) = \mathsf{GHASH}_H(N^{(b)})$ is found. This birthday collision attack can have a significantly higher success probability than the original attacks on the verification or decryption oracle.

Assume the polynomial $q(x)$ is chosen randomly and independently, and $H \neq 0$. The success probability for the original trial-and-error method querying the verification or decryption oracle is

$$n(l_N + 1)/2^{128}, \tag{3.1}$$

where n is the number of queries that have been made and l_N is the maximum number of blocks for nonces; while the lower bound for the success probability of the birthday attack is (see Lemma A.10 in Section A.4 of [71])

$$0.25\, n(n-1)(l_N + 1)/2^{128}. \tag{3.2}$$

In addition to the first encrypted counter blocks, we can also collect the following blocks into $\mathcal{S}$, in which way we may achieve even larger success probabilities. For example,

$E_K(N_i^{(a)})$ may be equal to $E_K(N_j^{(b)})$ for certain i and j. The upper bound of the collision probability for this case can be obtained from the polynomial (2.4) in Result 2.1, and the lower bound is the same as the polynomial (3.2). Although the success probability of this case is higher than the previous methods based on trial-and-error and birthday attacks, the collision $N_i^{(a)} = N_j^{(b)}$ may need more time complexity to be utilized for MAC forgery attacks. One naive way is to try every polynomial over the finite field that can be converted from inc^r with the specific r, and this will cost 2^{22} time at most.

Another benefit of attacking the encryption oracle is that, if certain countermeasures on the decryption or verification oracle are carried out, such as forbidding nonce reuse, the original attacks would fail or be detected, but the attacks on the encryption oracle will be unaffected.

A practical attack example on non-96-bit nonces is given as follows. We only give a basic example for this case, where the values and the polynomial $q(x)$ computed in the previous example are reused here.

Construct the polynomial $q'(x) = x^2 q(x)$, and apply $q'(x)$ to N to get a new 512-bit nonce N', i.e., $N' = (N_4 \oplus Q_3)||(N_3 \oplus Q_2)||(N_2 \oplus Q_1)||(N_1 \oplus Q_0)$.

N'	87e961e67784011f72faafd95b0eb640
	5bbceed48c991388a18f66e4ad62d270
	ef901d7ed10b1fd6963801d9083eebf9
	bd1bfa850ce25e8955588e8a50361ee5

Feeding (N', P) to the encryption oracle will result in the same ciphertext as C, so we are sure that the authentication H is in the set $\mathcal{H}$, and further MAC forgeries can be carried out by using $q(x)$.

3.5 Attacking GCM in the MAC-then-Enc Paradigm

GCM follows the Enc-then-MAC paradigm, i.e., authentication tags are computed based on ciphertexts. It is known that once the integrity of the system is compromised, the whole system including encrypted data will not be trustworthy. For GCM, if we successfully perform a MAC forgery attack described in previous sections, e.g., a forged tuple (N, C', T'), based on a valid (N, C, T), is fed to the decryption oracle and passes verification, the oracle will return P' that may have a simple linear difference with P. In this way, P can be obtained even without any knowledge of the encryption key. Therefore, the message authentication algorithm must be well protected.

One potential and straightforward option, which is indicated in [100], is to change GCM to a MAC-then-Enc scheme (MtE GCM, thereafter). More precisely, in MtE GCM, GMAC is computed based on plaintexts instead of ciphertexts, and the authentication tag is encrypted by block ciphers in the CTR mode.

However, we can show that the MAC forgery attacks described in the previous sections may still work on MtE GCM as these attacks are based on the linear properties of the polynomial-based MAC schemes. If no length extension is needed, applying $q(x)$ directly to ciphertexts and encrypted tags may successfully result in MAC forgeries. Consider the simplified case with

$$\begin{aligned} ET = &\ h_H(P) \oplus E_K(N) \oplus E_K(N_t) \\ = &\ h_H(P) \oplus \mathsf{Mask} \\ = &\ h_H(C \oplus S) \oplus \mathsf{Mask}, \end{aligned}$$

where ET is the encrypted authentication tag, $E_K(N_t)$ is to encrypt the authentication tag, $\mathsf{Mask} = E_K(N) \oplus E_K(N_t)$, S is the key stream produced by the CTR mode, and the other variables are the same as the ones in the previous sections. If we know a function $q(x)$ such that $q(H) = 0$, then

$$\begin{aligned} ET' = ET \oplus Q_0 = &\ h_H(C \oplus S) \oplus q^*(H) \oplus \mathsf{Mask} \\ = &\ g_{C \oplus S}(H) \oplus g_{Q^*}(H) \oplus \mathsf{Mask} \\ = &\ g_{C \oplus Q^* \oplus S}(H) \oplus \mathsf{Mask} \\ = &\ h_H(C \oplus Q^* \oplus S) \oplus \mathsf{Mask} \\ = &\ h_H(C' \oplus S) \oplus \mathsf{Mask}. \end{aligned}$$

This implies the tuple (N, C', ET'), where $C' = C \oplus Q^*$ and $ET' = ET \oplus Q_0$, will pass the verification oracle of MtE GCM.

A computation example is given as follows. The same K, N, H_1, and H_2 as in the previous examples are used. In order to avoid length extension, P is chosen to be longer, and H_3 is explicitly chosen to be $H_1 \cdot H_2/(H_1 \oplus H_2)$.

P	`705da82292143d2c949dc4ba014f6396`
	`705da82292143d2c949dc4ba014f6396`
C	`a51ccc6d2c45540cac4fde8b1e36802d`
	`a4bd55da5dcde1d763021d44f5fb3ab8`
ET	`5aba7c39516a4a90f738eaf61b02514a`
H_3	`6e0b0d1eaf109b0f26926be82780085c`

Constructing the polynomial $q(x)$, we can have its coefficients as follows.

Q_3	80000000000000000000000000000000
Q_2	bc7f3ddf8ee1d642f97ed862ed7fe40e
Q_1	00000000000000000000000000000000
Q_0	c52222258b2614c4c6f5981c65f15acd

Please note $Q_1 = 0$, so the length padding block in GHASH can stay unchanged.

The new ciphertext and encrypted authentication tag are $C' = (C_2 \oplus Q_3)||(C_1 \oplus Q_2)$ and $ET' = ET \oplus Q_0$.

C'	a51ccc6d2c45540cac4fde8b1e36802d
	a4bd55da5dcde1d763021d44f5fb3ab8
ET'	5aba7c39516a4a90f738eaf61b02514a

(N, C', ET') passes the verification oracle of MtE GCM.

If $\mathsf{len}(Q^*) > \mathsf{len}(C)$, i.e., the length extension is needed, the above attack on MtE GCM may not work. To decrypt $C \oplus Q^*$, where $\mathsf{len}(C \oplus Q^*) > \mathsf{len}(C)$, the verification oracle will produce longer key stream $S' = S||S_u$ with an unknown portion, S_u, so the output of the oracle will become unpredictable. However, adversaries may avoid this situation by trying to attack GHASH in the encryption oracle as discussed in Section 3.4, or simply waiting for longer ciphertexts.

Therefore, we can see that changing GCM into the MAC-then-Enc paradigm would add little strength against these MAC forgery attacks.

3.6 Summary

In this chapter, we have demonstrated that hash collisions are not necessarily required to construct successful MAC forgeries, and any polynomials can be used in these attacks. These new discoveries remove the restrictions in Procter and Cid's attacks. We have proven that all subsets of keys with no less than two elements are weak key classes for GCM-like polynomial-based MAC schemes, which is an extension to Procter and Cid's analysis. Moreover, we have presented a novel approach to transform these MAC forgery attacks into birthday attacks to increase their success probabilities in the case of GCM. The success probabilities of these attacks are summarized in Table 3.1. In addition, we have shown that these MAC forgeries attacks will still succeed if GCM is modified to the MAC-then-Enc paradigm, as one of the options mentioned in [100], such that authentication tags are protected by block cipher encryptions with the CTR mode.

Table 3.1: Success probabilities of the MAC forgery attacks on GCM.

Method	Success Probability	Reference
Trial-and-error	$n(l_N+1)/2^{128}$	[100]
Birthday attacks	$0.25\,n(n-1)(l_N+1)/2^{128}$	Section 3.4
Birthday attacks with inc	$\leq 2^{22}(n-1)(n+\sigma)(l_N+1)/2^{128}$	Section 3.4

As the discussion in Section 2.5, we further suggest that GCM may preferably be used with 96-bit nonces. Reusing GHASH in both generating initial counter numbers and computing authentication tags will help attackers to amplify their success probabilities for MAC forgeries as we have discussed in Section 3.4. For applications that prefer using non-96-bit nonces, e.g., negotiated data in a security protocol, we suggest applying the fix to GCM proposed in Section 2.3, i.e., using LGCM defined in Algorithm 2.2.

Chapter 4

Multidimensional Meet-in-the-Middle Attacks on Block Ciphers

Our work on meet-in-the-middle attacks on lightweight block ciphers is introduced in this chapter. We first describe the KATAN block cipher family and the original meet-in-the-middle attacks in Section 4.1. The general idea and framework of our multidimensional attacks are given in Section 4.2. Section 4.3 and Section 4.4 present new attacks on KATAN32/48/64 for real attack examples, and Section 4.5 discusses several potential optimization methods for improvements. In addition, we propose new attacks on KATAN with less numbers of rounds in Section 4.6, in order to demonstrate this new kind of attacks can be more time-efficient and memory-efficient than existing attacks. Finally, the last section summarizes the chapter.

4.1 Preliminaries

This section first briefly introduces the block cipher family, KATAN, and then presents a theoretical description for meet-in-the-middle (MITM, hereafter) attacks as the fundamential for subsequent sections.

4.1.1 The KATAN Family of Block Ciphers

KATAN is a lightweight block cipher family with efficient hardware performance and small software footprints [33]. It consists of three versions with different block sizes, 32, 48 and

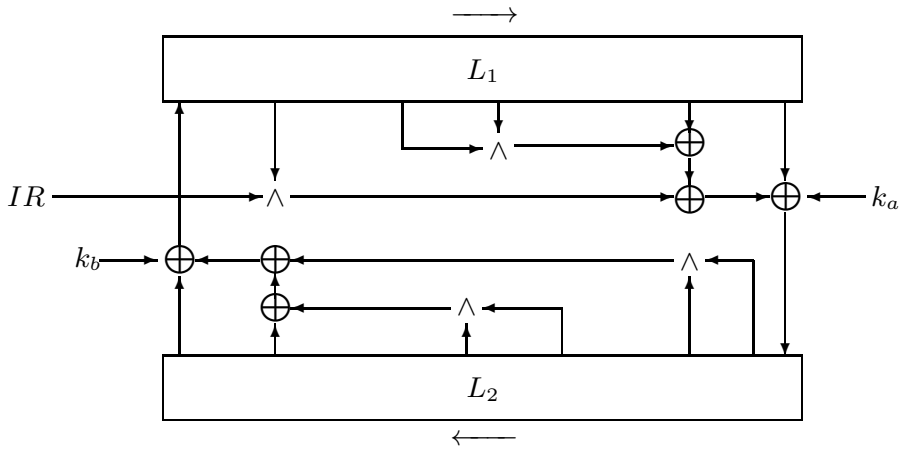

Figure 4.1: Structure of KATAN [33].

64 bits, which are named KATAN32, KATAN48 and KATAN64, respectively. Despite of the different block sizes, they all use 80-bit master keys. The structure of KATAN is shown in Figure 4.1, where $\wedge$ implies the bitwise AND.

In the encryption process of KATAN, the plaintext p is first divided to two pieces and loaded into the registers L_1 and L_2. Next, two nonlinear functions defined by the equations (4.1) are operated on L_1 and L_2 respectively.

$$\begin{aligned} f_a[L_1] = &\ L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR) \oplus k_a \\ f_b[L_2] = &\ L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_b \end{aligned} \tag{4.1}$$

In the above equations, x_i and y_j are predefined indices for different versions of KATAN, and IR is an irregular update sequence for preventing self-similarity attacks. The values of x_i, y_j and IR are given in Table 4.1 and Table 4.2, respectively.

Table 4.1: Parameters for KATAN.

Algorithms	$\|L_1\|$	$\|L_2\|$	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
KATAN32	13	19	12	7	8	5	3	18	7	12	10	8	3
KATAN48	19	29	18	12	15	7	6	28	19	21	13	15	6
KATAN64	25	39	24	15	20	11	9	38	25	33	21	14	9

k_a and k_b are two sub-key bits produced from a 80-bit master key K by a linear feedback shift register (LFSR). The master key K is loaded as the initial state of the LFSR, and

Table 4.2: The irregular update sequence IR for KATAN.

Round Index	IR
1 - 20	1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1
21 - 40	1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0
41 - 60	0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0
61 - 80	0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1
81 - 100	1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1
101 - 120	0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1
121 - 140	1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0
141 - 160	1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1
161 - 180	1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0
181 - 200	0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0
201 - 220	1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0
221 - 240	0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1
241 - 254	1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0

each output bit of the LFSR is used as a sub-key bit sequentially. Assume $\{k_i\}$ is the output sequence of the LFSR. For $0 \leq i \leq 79$, k_i is equal to the i-th bit of the master key; for $i \geq 80$, the sub-key bits can be computed by

$$k_i = k_{i-80} \oplus k_{i-61} \oplus k_{i-50} \oplus k_{i-13}. \tag{4.2}$$

For the r-th round of KATAN, $k_a = k_{2r-2}$ and $k_b = k_{2r-1}$, where $1 \leq r \leq 254$.

For KATAN32, after computing $f_a[L_1]$ and $f_b[L_2]$, the registers L_1 and L_2 are shifted to left by one bit, and the most significant bits of L_1 and L_2 are discarded. Next, $f_a[L_1]$ is fed into the least significant bit of L_2, and $f_b[L_2]$ is put into L_1. After 254 rounds of such operations, the states of L_1 and L_2 are concatenated and output as the ciphertext c.

KATAN48 and KATAN64 have the same structure and number of rounds as KATAN32, but L_1 and L_2 of KATAN48 and KATAN64 are updated two and three times in every round, respectively, by using the same k_a and k_b.

4.1.2 A Theoretical Description of Meet-in-the-Middle Attacks

We first take Double-DES (2DES) to explain the idea of MITM attacks. Let $c = \mathrm{DES}_k(p)$ denote one DES encryption, where k is the 56-bit master key, and p and c are the plaintext

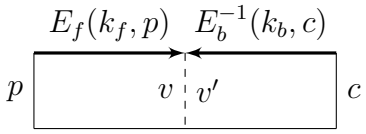

Figure 4.2: An illustration of meet-in-the-middle attacks.

and ciphertext, respectively. 2DES uses two different keys k_1 and k_2, and its encryption is computed as

$$c = \text{2DES}_{(k_1,k_2)}(p) = \text{DES}_{k_2}(\text{DES}_{k_1}(p)) \ .$$

The total number of key bits is $2 \cdot 56 = 112$, so the time complexity of the exhaustive key search for 2DES is 2^{112}.

To launch a MITM attack, firstly, we compute $v = \text{DES}_{k_1}(p)$ for all possible k_1's, and store all v's into a set S with corresponding k_1's. The time complexity of this step is 2^{56}. Secondly, from the ciphertext side, we compute the decryption $v' = \text{DES}_{k_2}^{-1}(c)$ for each possible k_2, and then check whether v' is in the set S. If we find a match, then the corresponding key pair (k_1, k_2) is possibly the correct one. This step needs to evaluate DES for $2 \cdot 2^{56} = 2^{57}$ times, which is much less than 2^{112}. This is the reason why we should use Triple-DES, rather than Double-DES, to obtain a reasonably larger security margin than DES.

More formally, assume that a cipher $c = E(k, p)$ can be decomposed into two consecutive sub-ciphers $E_f(k_f, \cdot)$ and $E_b(k_b, \cdot)$, i.e., $c = E_b(k_b, E_f(k_f, p))$, where k_f and k_b are the sub-keys used in E_f and E_b. Here f and b are the abbreviations for *forward* and *backward*. The steps of MITM attacks can be written as follows.

1. MITM phase, as shown in Figure 4.2:

   1.1 By iterating each possible k_f, compute the encryption $v = E_f(k_f, p)$, and collect v's into a set S.

   1.2 For every possible k_b, compute the decryption $v' = E_b^{-1}(k_b, c)$. Check whether $v' \in S$. If so, output the corresponding key pair (k_f, k_b) as a possibly correct key.

2. Brute-force testing phase:

   - If the MITM phase generates more than one pair of (k_f, k_b), then we need to use additional plaintext-ciphertext pairs to perform complete encryptions/decryptions to test them and find the correct one.

Abusing the notation, here we use $|\cdot|$ rather than $\mathsf{len}(\cdot)$ to represent the bit-length of a variable in order to simplify mathematical expressions and save space. n denotes the block size of a cipher, e.g., $n = |p| = |c|$. For simplicity, we assume bit-lengths of ciphers' intermediate states are smaller than block sizes, e.g., $|v| \leq n$, in the following content.

The time complexity for Step 1.1 is $2^{|k_f|}$, and for Step 1.2 it is $2^{|k_b|}$. During the MITM phase, wrong keys have the probability of $1/2^{|v|}$ to obtain a false positive. Thus if k_f and k_b do not have common key bits, the number of wrong keys passing the MITM phase will be $2^{|k_f|+|k_b|}/2^{|v|} = 2^{|k_f|+|k_b|-|v|}$. If k_f and k_b have common key bits, we let k_c denote all the key bits contained in both k_f and k_b, so the number of remaining keys will be $2^{|k_f|+|k_b|-|k_c|-|v|}$. We further assume k is the master key that consists of all the key bits of k_f and k_b, and $|v|$ is equal to the block size n. Then we have

$$2^{|k_f|+|k_b|-|k_c|-|v|} = 2^{|k|-|v|} = 2^{|k|-n}.$$

This can be easily understood from the information theory's point of view: Since we have the information of an n-bit plaintext-ciphertext pair (p, c), we can only reduce the key space to $1/2^n$ of the original. After this MITM phase, we can simply employ brute-force testing to remove wrong keys.

The time complexity of the first attempt of brute-force testing will be equal to $2^{|k|-n}$. The probability of wrong keys passing the testing is $1/2^n$ on average, so $2^{|k|-2n}$ keys will pass the first testing. If $2^{|k|-2n}$ is still larger than 1, we can use another plaintext-ciphertext pair to perform additional testing to further reduce the key space. The overall time complexity of the brute-force testing phase will be $2^{|k|-n} + 2^{|k|-2n} + 2^{|k|-3n} + \cdots$, and this phase needs $\lceil (|k| - n)/n \rceil$ pairs of plaintexts and ciphertexts.

To sum up, the total time complexity of the MITM attack is

$$2^{|k_f|} + 2^{|k_b|} + 2^{|k|-n} + 2^{|k|-2n} + 2^{|k|-3n} + \cdots \approx 2^{|k_f|} + 2^{|k_b|} + 2^{|k|-n} \ ,$$

and the total data complexity is $\lceil (|k| - n)/n \rceil + 1 = \lceil |k|/n \rceil$. Similar analysis can be found in [29].

When a matching key pair (k_f, k_b) is found, it can be tested instantly, so we do not need to save it in memory and wait for other candidate keys. Therefore, the major memory consumption of this attack comes from maintaining the set S. There are many kinds of data structures to construct S, such as hash tables. Actually, the construction and look-up algorithms of S also have influence on the overall attack time. The look-up time is generally omitted since it is usually much less than a complete cipher encryption. We suggest using tables whose indices are (parts of) matching values, e.g., v in the above example, and

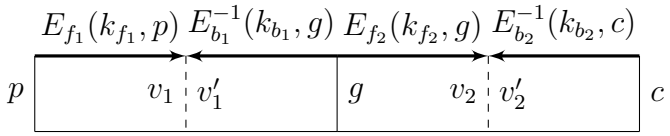

Figure 4.3: Meet-in-the-middle attacks with one guess.

letting each entry in the tables point to a (linked) list of corresponding sub-keys. Despite of different constructions of S, the memory complexity of the MITM attack should be at least $2^{|k_f|}$ or $2^{|k_b|}$.

4.2 A Framework for Multidimensional MITM Attacks

When designing lightweight block ciphers, e.g., for environment-constrained devices, we usually prefer to adopt small block sizes for efficient performance. However, due to security requirements, master keys cannot be too short. This usually leads us to cipher designs with key sizes larger than block sizes. Although this kind of designs is perfectly valid, it gives us the possibility to guess certain short intermediate states and divide the ciphers into small sub-ciphers for easier analysis.

Let us first give a simple and *inefficient* attack framework for ease of understanding. In the subsequent sections, refined methods are given in the attacks on the KATAN block cipher family.

Suppose we first guess an intermediate state g, as shown in Figure 4.3, and perform two MITM attacks on the sub-ciphers divided by g. By assuming a simplest case that the sub-keys k_{f_1}, k_{b_1}, k_{f_2} and k_{b_2} do not have common key bits, the attack steps can be described as follows.

1. Compute $v_1 = E_{f_1}(k_{f_1}, p)$ for each possible k_{f_1}, and put all k_{f_1}'s into a table T_1 indexed by v_1, each entry of which is a set of certain k_{f_1}'s.
2. Compute $v_2' = E_{b_2}^{-1}(k_{b_2}, c)$ for each possible k_{b_2}, and put all k_{b_2}'s into a table T_2' (similar as T_1) indexed by v_2'.
3. For each possible guess of g:
   (a) Compute $v_1' = E_{b_1}^{-1}(k_{b_1}, g)$ for each possible k_{b_1}, and maintain a table T_1' of k_{b_1} indexed by v_1'.

(b) Compute $v_2 = E_{f_2}(k_{f_2}, g)$ for each possible k_{f_2}, and maintain a table T_2 of k_{f_2} indexed by v_2.

(c) Every matching pair (k_{f_1}, k_{b_1}) for $v_1 = v_1'$, together with each matching pair (k_{f_2}, k_{b_2}) for $v_2 = v_2'$, forms a candidate key for the whole encryption. We use additional plaintext-ciphertext pairs to perform brute-force testing on these candidate keys. If one key passes all tests, then output it as the correct key.

Since we do not need to recompute E_{f_1} and $E_{b_2}^{-1}$ for different g's, the time complexity of this attack without the brute-force testing phase is

$$2^{|k_{f_1}|} + 2^{|k_{b_2}|} + 2^{|g|} \cdot (2^{|k_{b_1}|} + 2^{|k_{f_2}|}).$$

For each guessed value of g, the MITM step from p to g will reduce the size of the key space to $2^{|k|-|v_1|}$ and the second MITM step for the interval between g and c will further reduce it to $2^{|k|-|v_1|-|v_2|}$, so after the two MITM attacks the total number of remaining keys is $2^{|g|} \cdot (2^{|k|-|v_1|-|v_2|}) = 2^{|k|+|g|-|v_1|-|v_2|}$. Assuming $|g| = |v_1| = |v_2| = n$, we have $2^{|k|+|g|-|v_1|-|v_2|} = 2^{|k|-n}$, which is consistent with the analysis for original MITM attacks in the last section. The total time complexity of the brute-force step is still about $2^{|k|-n}$.

Please note a subtle part in the above analysis: Although the size of the master key space is reduced to $2^{|k|-|v_1|}$ after the MITM attack step from p to g, the number of sub-keys to be matched with the results from the other MITM step between g and c is only $2^{|k_{f_1}|+|k_{b_1}|-|v_1|}$, which may be much less than $2^{|k|-|v_1|}$.

The memory complexity of the attack is $2^{|k_{f_1}|} + 2^{|k_{b_1}|} + 2^{|k_{f_2}|} + 2^{|k_{b_2}|}$, since we may need to store T_1, T_1', T_2 and T_2' in memory. The data complexity of the attack is $\lceil |k|/n \rceil$.

In general cases, the sub-keys, k_{f_1}, k_{b_1}, k_{f_2} and k_{b_2}, would involve many common key bits, so the above attack cannot be applied directly. A straightforward way to solve this is treating each sub-key bit as an independent new variable. This technique has been used in other cryptanalysis methods, such as [67]. But we may get more efficient results or attack more rounds by carefully investigating ciphers' detailed designs. For example, we may perform linear transformations before matching sub-keys, or study round functions to perform partial encryptions/decryptions. We show real attack examples using these techniques in following sections.

Certainly, we can guess more intermediate states and then segment ciphers into smaller pieces. MITM attacks with multiple guesses are illustrated in Figure 4.4. For simplicity of description, hereafter we denote the MITM attacks with multiple guesses as multidimensional MITM (MD-MITM) attacks, and especially the attacks with n sub-ciphers will be

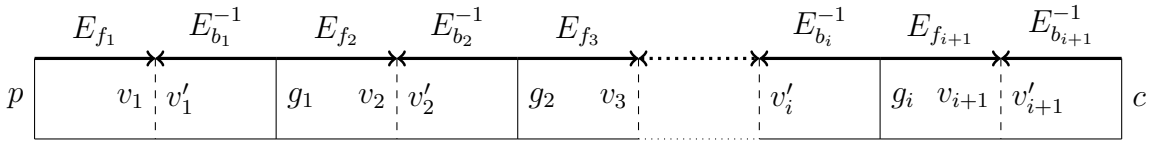

Figure 4.4: General process of meet-in-the-middle attacks with multiple guesses.

nD-MITM. For example, the above attack with one guess is a 2D-MITM attack, and the traditional MITM attacks can be viewed as 1D-MITM attacks.

The steps of an $(i+1)$D-MITM attack can be briefly stated as follows.

1. Construct a table T_1 of k_{f_1} by computing $v_1 = E_{f_1}(k_{f_1}, p)$.
2. Construct a table T'_{i+1} of $k_{b_{i+1}}$ by computing $v'_{i+1} = E^{-1}_{b_{i+1}}(k_{b_{i+1}}, c)$.
3. For each guess of g_1:
   (a) Construct a table T'_1 by computing $v'_1 = E^{-1}_{b_1}(k_{b_1}, g_1)$, which is to match with T_1.
   (b) Construct a table T_2 by computing $v_2 = E_{f_2}(k_{f_2}, g_1)$.
   (c) For each guess of g_2:
      i. Construct a table T'_2 by computing $v'_2 = E^{-1}_{b_2}(k_{b_2}, g_2)$, to match with T_2.
      ii. $\cdots$ (Perform recursive operations till g_i.)
      iii. For each guess of g_i:
         A. Construct a table T'_i by computing $v'_i = E^{-1}_{b_i}(k_{b_i}, g_i)$, to match with T_i.
         B. Compute $v_{i+1} = E_{f_{i+1}}(k_{f_{i+1}}, g_i)$, which can form a table T_{i+1} in order to match with T'_{i+1}.
         C. Perform brute-force testing on each combination of matching sub-key pairs from (T_1, T'_1), (T_2, T'_2), $\cdots$, (T_{i+1}, T'_{i+1}), and output the passing combination as the correct key.

If we assume $|g_1| = |g_2| = \cdots = n$, the time complexity of the MITM phase with multiple guesses is

$$\begin{aligned} & 2^{|k_{f_1}|} + 2^{|k_{b_{i+1}}|} + 2^{|g_1|} \cdot (2^{|k_{b_1}|} + 2^{|k_{f_2}|} + 2^{|g_2|} \cdot (2^{|k_{b_2}|} + 2^{|k_{f_3}|} + \cdots + 2^{|g_i|} \cdot (2^{|k_{b_i}|} + 2^{|k_{f_{i+1}}|}))) \\ = \; & 2^{|k_{f_1}|} + 2^{|k_{b_{i+1}}|} + 2^n \cdot (2^{|k_{b_1}|} + 2^{|k_{f_2}|}) + 2^{2n} \cdot (2^{|k_{b_2}|} + 2^{|k_{f_3}|}) + \cdots + 2^{i \cdot n} \cdot (2^{|k_{b_i}|} + 2^{|k_{f_{i+1}}|}). \end{aligned} \tag{4.3}$$

Please note that the order of $g_1, g_2, \cdots, g_i$ does not matter, and we can first guess any ones of them in order to obtain more desirable results.

The memory complexities of MD-MITM attacks are upper-bounded by the total memory consumption of $T_1, T_1', T_2, \cdots, T_{i+1}'$. In order to obtain the minimum value for the time complexity equation (4.3), the sizes of $T_1', T_2, \cdots, T_{i+1}$, i.e., $2^{|k_{b_1}|}, 2^{|k_{f_2}|}, \cdots, 2^{|k_{f_{n+1}}|}$, should be much smaller than the sizes of T_1 and T_{i+1}', i.e., $2^{|k_{f_1}|}$ and $2^{|k_{b_{i+1}}|}$. So it is safe to ignore $T_1', T_2, \cdots, T_{i+1}$ here and give a simple upper bound for the memory complexities of these MD-MITM attacks as $2^{|k_{f_1}|} + 2^{|k_{b_{i+1}}|}$.

We only use one known plaintext-ciphertext pair before the brute-force testing phase, so it is easy to see that MD-MITM attacks have the same data complexities as 1D-MITM attacks, i.e., $\lceil |k|/n \rceil$ known plaintext-ciphertext pairs.

What we want to emphasize here is that the theoretical analysis in this section only presents a general framework to perform the divide-and-conquer method on ciphers. Whether it can really work and improve attacks' efficiencies on a specific cipher depends on the detailed design of the cipher. In subsequent sections, we present several MD-MITM attacks on the KATAN block cipher family, which can attack more rounds and reduce complexities of previous attacks. These new attacks highly rely on certain design details of KATAN.

We also want to note that there is an independent work [42] with similar ideas, but the authors focus on optimizing time-memory trade-offs for composite problems, and their analysis is only applied to the cases where all sub-ciphers have independent keys.

4.3 MD-MITM Attacks on KATAN32

The section applies the multidimensional framework to reduced-round KATAN32, and proposes new attacks that can break more numbers of rounds than the existing attacks. For ease of understanding, we first describe a simple attack procedure and then improve upon it.

4.3.1 2D-MITM Attacks on KATAN32

Use s_i to denote the intermediate state right after the i-th round encryption of KATAN32/48/64, which implies $s_0 = p$ and $s_{254} = c$. We first show a simplest case of 2D-MITM on KATAN32, and improve it in subsequent discussions. For simplicity, we use $E_i(s)$ to denote $E_{f_i}(k_{f_i}, s)$,

and $D_j(s)$ to denote $E_{b_j}^{-1}(k_{b_j}, s)$. $k_{i...j}$ is the sub-key containing all the sub-key bits whose indices are from i to j. The attack procedure, which is a standard 2D-MITM attack, is as follows.

1. Compute $s_{40} = E_1(s_0)$ by using every possible $k_{f_1} = k_{0...79}$, and compute $k_{80...127}$ from k_{f_1} by using linear functions derived from the LFSR (see the equation (4.2)). Put each k_{f_1} in a table T_1 indexed by s_{40} and $k_{80...127}$. Each entry of the table should have one element on average.

2. Compute $s'_{88} = D_2(s_{128})$ by using every possible $k_{b_2} = k_{176...255}$, and compute $k_{128...175}$ from k_{b_2} by using linear functions derived from the LFSR. Store each k_{b_2} in a table T'_2 indexed by s'_{88} and $k_{128...175}$. Similarly, each entry of the table has one element on average.

3. For each guess of $g = s_{64}$:

   (a) Compute $s'_{40} = D_1(g)$ for each $k_{b_1} = k_{80...127}$, and then find the matching key k_{f_1} in T_1. On average there is only one matching key, and we put it in a set S.

   (b) Compute $s_{88} = E_2(g)$ for each $k_{f_2} = k_{128...175}$, and find the matching key k_{b_2} in T'_2. Then compute $k_{0...79}$ from $k_{b_2} = k_{176...255}$ by using linear equations derived from the LFSR, and check whether it is in the set S. If so, perform brute-force testing on this candidate key. If it passes all tests, output it as the correct master key.

To fairly compare with the time complexities of existing attacks, we adopt the formula proposed in [120] to estimate the time complexity of the above attack on KATAN32. Use R_{f_1}, R_{b_1}, R_{f_2} and R_{b_2} to denote the numbers of rounds involved in the different phases of the 2D-MITM attack, and R to denote the total number of the attacked rounds. The time complexity without the brute-force testing phase is computed as follows.

$$2^{|k_{f_1}|} \cdot \frac{R_{f_1}}{R} + 2^{|k_{b_2}|} \cdot \frac{R_{b_2}}{R} + 2^n \cdot \left(2^{|k_{b_1}|} \cdot \frac{R_{b_1}}{R} + 2^{|k_{f_2}|} \cdot \frac{R_{f_2}}{R} \right)$$

The above equation can be seen as estimating the equivalent number of full R-round cipher evaluations, as the time to complete the 2D-MITM attack.

Here we simply ignore the time complexities of linear transformations when matching sub-keys, because these LFSR computations only involve several linear operations, which are more cost-efficient compared with iterations of nonlinear round functions, and exhaustive key search also needs to compute the sub-keys and will consume the equivalent time.

In the above 2D-MITM attack on KATAN32, since $|k_{f_1}| = |k_{b_2}| = 80$ and $|k_{b_1}| = |k_{f_2}| = 48$, its total time complexity is

$$2^{80} \cdot \frac{40}{128} + 2^{80} \cdot \frac{40}{128} + 2^{32} \cdot \left(2^{48} \cdot \frac{24}{128} + 2^{48} \cdot \frac{24}{128} \right) + 2^{80-32} \approx 2^{80},$$

where 2^{80-32} is the time complexity of the brute-force testing phase. Please note that 2^{80} is exactly the time complexity of the exhaustive key search on KATAN. The memory complexity of the attack is $2^{80} + 2^{80} + 2^{80-32} \approx 2^{81}$, since we need to store T_1, T_2 and S in memory. The data complexity is still the same as 1D-MITM attacks, i.e., $\lceil 80/32 \rceil = 3$ plaintext-ciphertext pairs.

Reducing Time Complexities

In order to make the time complexity of the above attack better than exhaustive search, i.e., 2^{80}, we can reduce the numbers of attacked rounds in the first forward and second backward phases by one. But in this case, when constructing T_1 (or similarly T_2'), we cannot simply use the intermediate state $g = s_{38}$ and $k_{b_1} = k_{78...125}$ as indices like in the previous 2D-MITM attack, because the new $k_{f_1} = k_{0...77}$ does not have the full 80-bit information of the master key $K = k_{0...79}$, and certain bits of k_{b_1} still depend on the values of k_{78} and k_{79}.

However, by assuming k_{78} and k_{79} are zero, we can compute a temporal key $\kappa_{80...125}$ for the purpose of matching, which is equivalent to removing these two bits from the linear equations of key scheduling. Under such circumstance, the first MITM phase between p and g can be performed without knowing k_{78} and k_{79}, and later extra information of k_{78} and k_{79} can be appended to form the 80-bit master key. The detailed attack procedure is described as follows.

1. Compute $s_{39} = E_1(s_0)$ by iterating every possible $k_{f_1} = k_{0...77}$, and compute $\kappa_{80...125}$ from $k_{0...77}$ by assuming $k_{78} = k_{79} = 0$. Put each k_{f_1} in a table T_1 indexed by s_{39} and $\kappa_{80...125}$. Each entry of the table should have one element on average.

2. Compute $s'_{87} = D_2(s_{126})$ by using every possible $k_{b_2} = k_{174...251}$, and compute $\kappa_{126...171}$ from $k_{174...251}$ by treating $k_{172} = k_{173} = 0$. Store each k_{b_2} in a table T_2' indexed by s'_{87} and $\kappa_{126...171}$. Similarly, each entry of the table has one element on average.

3. For each guess of $g = s_{63}$:

(a) Compute $s'_{39} = D_1(g)$ for each $k_{b_1} = k_{78...125}$, and compute $\kappa_{80...125}$ by subtracting k_{78} and k_{79} from the bits of $k_{80...125}$. Then use s'_{39} and $\kappa_{80...125}$ to find the matching k_{f_1} in T_1. On average there is only one matching $k_{f_1} = k_{0...77}$, so we combine it with the k_{78} and k_{79} to form a candidate master key. Put the candidate master key in a set S.

(b) Compute $s_{87} = E_2(g)$ for each $k_{f_2} = k_{126...173}$, and compute $\kappa_{126...171}$ by subtracting k_{172} and k_{173} from $k_{126...171}$. Then use s_{87} and $\kappa_{126...171}$ to find the matching $k_{b_2} = k_{174...251}$ in T'_2. Compute $k_{0...79}$ from $k_{172...251}$ by using linear functions derived from the LFSR, and check whether it is in the set S. If so, perform brute-force testing on it.

The overall time complexity of this attack, on 126-round KATAN32, is

$$2^{78} \cdot \frac{39}{126} + 2^{78} \cdot \frac{39}{126} + 2^{32} \cdot \left(2^{48} \cdot \frac{24}{126} + 2^{48} \cdot \frac{24}{126}\right) + 2^{80-32} \approx 2^{79.10},$$

and its memory complexity is $2^{78} + 2^{78} + 2^{48} \approx 2^{79}$. This attack has already reached more rounds than any existing attack on KATAN32, but we still have room for improvements.

Increasing the Number of Attacked Rounds

We can see there is a large time complexity gap between the MD-MITM and brute-force testing phases in the above 2D-MITM attacks on KATAN32. The time complexities of brute-force testing are always $2^{80-32} = 2^{48}$, and the complexities of MITM phases are close to 2^{80}. If we can balance complexities of the two phases, the overall time complexities may drop. One way to achieve this is reducing bit-lengths of the intermediate states for matching, i.e., $v_1, v_2, \cdots, v_{i+1}$. As a result, computing incomplete parts of original $v_1, v_2, \cdots, v_{i+1}$ may not need to use up all of the sub-key bits, and we can extend the attack to more rounds. In return, more candidate keys are left to be tested in brute-force phases. This technique is called *partial matching*, and has been used in various papers, such as [29, 120].

By adopting the partial matching technique, we can extend our attack on 126 rounds of KATAN32 to 152 rounds. For simplicity, we only use the partial matching technique in the second MITM part. After searching all possible combinations of intermediate states by programs, we find the best position for the second backward phase is from s_{152} to s_{87}. Based on the 78-bit information of $k_{b_2} = k_{226...303}$, we can still compute 2 bits of s_{87}. By

using these 2 bits for matching, there will be 2^{78} candidate keys left for brute-force testing, and thus the total time complexity of the attack should be still less than 2^{80}.

The partial-matching details are shown as follows. The column a is for k_a, and b is for k_b. Here we use the same notations as [29] and [120]: *0* implies this bit is fully computable based on information we know from one side of a MITM attack, and thus considered as *known*; *1* means computing this bit needs extra key information from the opposite side of the MITM attack, and is considered as *unknown*. To form a matching, the two resultant bits from both sides should be *known*.

Rd.	a	b	L1	L2
second backward phase				
114	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
113	1	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
112	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
111	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
110	0	0	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
109	0	0	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
108	0	0	0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
107	0	0	0 0 0 0 0 0 0 0 1 0 0 0 1 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
106	0	0	0 0 0 0 0 0 0 1 0 0 0 1 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
105	0	0	0 0 0 0 0 0 1 0 0 0 1 1 0 1 1	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
104	1	0	0 0 0 0 1 0 0 0 1 1 0 1 1 1	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0
103	0	1	0 0 0 1 0 0 0 1 1 0 1 1 1 1	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1
102	0	0	0 0 1 0 0 0 0 1 1 0 1 1 1 1	0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0
101	0	0	1 0 0 0 0 1 1 0 1 1 1 1 1 1	0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0
100	0	0	0 0 0 0 1 1 0 1 1 1 1 1 1 1	0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1
99	0	0	0 0 0 1 1 0 1 1 1 1 1 1 1 1	0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1
98	1	1	0 0 1 1 0 1 1 1 1 1 1 1 1 1	0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 1
97	0	0	1 1 0 1 1 1 1 1 1 1 1 1 1 1	0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 1 1
96	0	0	1 0 1 1 1 1 1 1 1 1 1 1 1 1	0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 1 1 1
95	0	0	0 1 1 1 1 1 1 1 1 1 1 1 1 1	0 1 0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1
94	1	1	1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1
93	0	0	1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1
92	0	0	1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1
91	0	0	1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1
90	0	0	1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1
89	1	0	1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
88	0	1	1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
second forward phase				
87	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
matching				
2 bits			1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

For the second MITM attack with partial matching, we cannot simply construct T_2' by using the 2 matching bits, along with a 46-bit temporal key computed from k_{b_2}, as indices.

Therefore, we propose using a *product set* that is constructed by two related tables: First find an index value from the first table, and then locate the target value from the second table by using the index value. The attack procedure is described as follows.

1. For every possible $k_{f_1} = k_{0...77}$, compute $s_{39} = E_1(s_0)$, and calculate $\kappa_{80...125}$ by treating $k_{78} = k_{79} = 0$. Save k_{f_1} in a table indexed by s_{39} and $\kappa_{80...125}$. This step is similar to the previous attacks. Every entry of the table will have one element on average.

2. Compute the 2 bits of $s'_{87} = D_2(s_{152})$ by using every possible $k_{b_2} = k_{226...303}$, and save all computation results in a table T'_2, whose index is k_{b_2}.

3. For each guess of $g = s_{63}$:

   (a) Compute the 2 bits of $s_{87} = E_2(g)$ for every possible $k_{f_2} = k_{126...173}$, and store all k_{f_2}'s in a table T_2 indexed by different values of the 2 bits. Each entry of T_2 is a sub-set of k_{f_2}'s. After this step, T_2 and T'_2 together form a *product set*, and we show how to look up its elements in the next step.

   (b) For each $k_{b_1} = k_{78...125}$:

      i. Compute $s'_{39} = D_1(g)$. Calculate $\kappa_{80...125}$ from $k_{80...125}$ by subtracting k_{78} and k_{79}. Use s'_{39} and $\kappa_{80...125}$ to find the matching k_{f_1} in T_1. Now we have the (guessed) full 80-bit information of $k_{0...79}$.

      ii. Based on the knowledge of $k_{0...79}$, compute the sub-key pair k_{f_2} and k_{b_2}, and check whether the pair is also in the product set of T_2 and T'_2: First look up k_{b_2} in T'_2 to find the corresponding values of the two bits, and then check whether k_{f_2} is in the entry (set) of T_2 indexed by the two bits. If so, perform further brute-force testing on the candidate key.

To further explain the computation of the temporal sub-keys, we take $\kappa_{80...125}$ in Step 3(b)(i) for example: Since each bit of $k_{80...125}$ can be expressed by a linear expression in terms of the bits of $k_{0...79}$, we can subtract the values of k_{78} and k_{79} from these linear expressions to get a temporal value, i.e., $\kappa_{80...125}$, only for the purpose of matching. Consequently, we can remove wrong candidate key pairs by using the knowledge of the key bits except k_{78} and k_{79}.

The total time complexity of this 2D-MITM attack is

$$2^{78} \cdot \frac{39}{152} + 2^{78} \cdot \frac{65}{152} + 2^{32} \cdot \left(2^{48} \cdot \frac{24}{152} + 2^{48} \cdot \frac{24}{152} \right) + 2^{80-2} \approx 2^{79.56}.$$

The memory consumption includes the storage for T_1, T_2 and T_2', and the total is the same as the previous attack, i.e., 2^{79}. Please note that the data complexity of this attack is also the same as before, i.e., 3 known plaintext-ciphertext pairs, because even if the first pair used in the MITM phase is only consumed by 2-bit information, we can reuse it in the brute-force testing phase to filter out more wrong keys.

4.3.2 3D-MITM Attacks on KATAN32

For MD-MITM attacks, the computations of certain steps may have been repeated. For example, as in Figure 4.4, E_{f_3} is computed for 2^{2n} times, so we may cache the computation results of first 2^n times and reuse them later.

As the previous subsection, we first start from a simple attack. The two guessed states are $g_1 = s_{64}$ and $g_2 = s_{88}$. The first MITM part starts from s_0, ends at g_1, and meets at s_{40}. The second MITM part starts from g_1, ends at g_2, and meets at s_{80}. The third one is from g_2 to s_{152}, and meets at s_{112}. A subtle part of this attack is keeping the numbers of the key bits of k_{f_2} and k_{b_2} small enough, such that we expect only one sub-key in each intermediate matching step, in order to keep the attack simple. The detailed attack procedure is described as follows.

1. Compute $s_{40} = E_1(s_0)$ and $k_{b_1} = k_{80...127}$ for each possible $k_{f_1} = k_{0...79}$, and store k_{f_1} in a table T_1 indexed by s_{40} and k_{b_1}. Every entry of T_1 will have one element on average.

2. Compute $s'_{112} = D_3(s_{152})$ for each $k_{b_3} = k_{224...303}$, and store k_{b_3} in a table T_3' indexed s'_{112}. Each entry of T_3' is a set containing certain k_{b_3}'s.

3. For each guessed pair of $g_2 = s_{88}$ and $k_{f_3} = k_{176...223}$, compute $s_{112} = E_3(g_2)$ and store the computation results in a table T_3 indexed by g_2 and k_{f_3}. After this step, T_3 and T_3' form a product set.

4. For each guess of $g_1 = s_{64}$:

   (a) Compute $s'_{40} = D_1(g_1)$ for each $k_{b_1} = k_{80...127}$, and find the matching $k_{f_1} = k_{0...79}$ in T_1 by using the indices k_{b_1} and s'_{40}. Next, based on the 80-bit master key $k_{0...79}$, we compute $k_{128...175}$, and then store $k_{0...79}$ in a table S indexed by $k_{128...175}$. Each entry of S will have one element on average.

   (b) Compute $s_{80} = E_2(g_1)$ for each 32-bit $k_{f_2} = k_{128...159}$, and store k_{f_2} in a table T_2 indexed by s_{80}, where each entry has one element on average.

(c) For each guess of $g_2 = s_{88}$:

- For each 16-bit $k_{b_2} = k_{160...175}$:
  i. Compute $s'_{80} = D_2(s_{88})$, and look up the table T_2 by s'_{80} to find the matching $k_{f_2} = k_{128...159}$. Thus we get a consecutive 48-bit sub-key $k_{128...175}$.
  ii. Look up the table S by $k_{128...175}$ to find the matching $k_{0...79}$.
  iii. Finally use $k_{0...79}$ to compute the corresponding pair of k_{f_3} and k_{b_3}, and check whether the pair is also in the product set of T_3 and T'_3. If so, do further brute-force testing on $k_{0...79}$.

The total number of attacked rounds is 152. The time complexity of this 3D-MITM attack is

$$2^{80} \cdot \tfrac{40}{152} + 2^{80} \cdot \tfrac{40}{152} + 2^{32+48} \cdot \tfrac{24}{152} + 2^{32} \cdot \left(2^{48} \cdot \tfrac{24}{152} + 2^{32} \cdot \tfrac{16}{152} + 2^{32+16} \cdot \tfrac{8}{152}\right) + 2^{80-32} \approx 2^{79.84}.$$

Since we need to store T_1, T_3, T'_3, S and T_2 in memory, the memory complexity is $2^{80} + 2^{80} + 2^{32+48} + 2^{48} + 2^{32} \approx 2^{81.58}$. We use only one known plaintext-ciphertext pair in the MITM attack phase, so the total data complexity of the 3D-MITM attack is 3 known plaintext-ciphertext pairs as before.

Further Improvements

To make the memory complexity become under 2^{80}, we can lower the numbers of the attacked rounds in the phases f_1, f_3 and b_3 by one. In this case, its memory complexity will decrease to about $2^{81.58-2} = 2^{79.58}$, and the time complexity will decrease as well.

The partial matching technique can also be used in 3D-MITM attacks to increase the number of attacked rounds. Adopting partial matching in the phases f_3 and b_3, we can still use the similar positions for the two matching bits as the 2D-MITM attack in Section 4.3.1.

Our final 3D-MITM attack on KATAN32 can reach 175 rounds at most. The first MITM part starts from s_0, meets at s_{39}, and ends at $g_1 = s_{63}$. The second one is from g_1 to $g_2 = s_{87}$, and meets at s_{79}. The third one meets at s_{110} and ends at s_{175}. The overall attack is similar to the previous 3M-MITM one, except partial matching is employed in the third MITM part. The detailed attack procedure is described as follows.

1. For each possible $k_{f_1} = k_{0...77}$, calculate $s_{39} = E_1(s_0)$, and compute $\kappa_{80...125}$ by treating $k_{78} = k_{79} = 0$. Store k_{f_1} in a table T_1 indexed by s_{39} and $\kappa_{80...125}$. Every entry of T_1 will have one element on average.

2. For each $k_{b_3} = k_{272...349}$, compute the 2 bits of $s'_{110} = D_3(s_{175})$, and store k_{b_3} in a table T'_3 indexed by values of the two bits. Each entry of T'_3 is a set containing certain k_{b_3}'s.

3. For each guessed pair of $g_2 = s_{87}$ and $k_{f_3} = k_{174...219}$, compute the two bits of $s_{110} = E_3(s_{87})$ and store the computation results in a table T_3 indexed by g_2 and k_{f_3}. After this step, T_3 and T'_3 form a product set.

4. For each guess of $g_1 = s_{63}$:

   (a) Compute $s'_{39} = D_1(g_1)$ for each $k_{b_1} = k_{78...125}$. Compute $\kappa_{80...125}$ from $k_{80...125}$ by subtracting k_{78} and k_{79}. Find the matching k_{f_1} in T_1 by using the indices $\kappa_{80...125}$ and s'_{39}. Next, compute $k_{126...173}$ based on $k_{0...79}$, and then store $k_{0...79}$ in a table S indexed by $k_{126...173}$. Each entry of S will have one element on average.

   (b) Compute $s_{79} = E_2(g_1)$ for each $k_{f_2} = k_{126...157}$, and store k_{f_2} in a table T_2 indexed by s_{79}, each entry of which has one element on average.

   (c) For each guess of $g_2 = s_{87}$:

   - For each 16-bit $k_{b_2} = k_{158...173}$:
     i. Compute $s'_{79} = D_2(g_2)$, and look up the table T_2 by s'_{79} to find the matching $k_{f_2} = k_{126...157}$. Now we have the complete 48-bit information about $k_{126...173}$.
     ii. Look up the table S by $k_{126...173}$ to find the matching $k_{0...79}$.
     iii. Finally use $k_{0...79}$ to compute the sub-key pair of k_{f_3} and k_{b_3}, and check whether the pair is also in the product set of T_3 and T'_3. If so, do further brute-force testing on $k_{0...79}$.

The total time complexity of this attack is

$$2^{78} \cdot \tfrac{39}{175} + 2^{78} \cdot \tfrac{65}{175} + 2^{32+46} \cdot \tfrac{23}{175} + 2^{32} \cdot \left(2^{48} \cdot \tfrac{24}{175} + 2^{32} \cdot \tfrac{16}{175} + 2^{32+16} \cdot \tfrac{8}{175}\right) + 2^{80-2} \approx 2^{79.30}.$$

The memory complexity is $2^{78} + 2^{78} + 2^{32+46} + 2^{48} + 2^{32} \approx 2^{79.58}$. The data complexity stays as the same, i.e., 3 known plaintext-ciphertext pairs.

4.4 MD-MITM Attacks on KATAN48 and KATAN64

We can apply similar MD-MITM attacks to other versions of KATAN, i.e., KATAN48 and KATAN64, but we can guess only one intermediate state g, because the block sizes of KATAN48 and KATAN64 are larger than halves of their key lengths. We can also use the partial matching technique here in order to increase the numbers of attacked rounds.

4.4.1 A 2D-MITM Attack on KATAN48

Our 2D-MITM attack on KATAN48 can reach 130 rounds at most. The guessed state is $g = s_{55}$. The first MITM part meets at s_{39}, and the second meets at s_{71}. The partial matching technique is used in the second MITM part. The attack steps are described as follows.

1. Compute $s_{39} = E_1(s_0)$ by using every possible $k_{f_1} = k_{0...77}$, and compute $\kappa_{80...109}$ by treating $k_{78} = k_{79} = 0$. Save $k_{0...77}$ in a table indexed by s_{39} and $\kappa_{80...125}$.
2. Compute the 2 bits of $s'_{71} = D_2(s_{130})$ by using every possible $k_{b_2} = k_{182...259}$, and save all computation results in a table T'_2, whose index is k_{b_2}.
3. For each guess of $g = s_{55}$:
   (a) Compute the 2 bits of $s_{71} = E_2(g)$ by using every possible $k_{f_2} = k_{110...141}$. Store all k_{f_2}'s in a table T_2 indexed by values of the 2 bits, and each entry is a sub-set containing certain k_{f_2}'s. After this step, T_2 and T'_2 together form a product set.
   (b) Compute $s'_{39} = D_1(g)$ for each $k_{b_1} = k_{78...109}$, calculate $\kappa_{80...109}$ by subtracting k_{78} and k_{79} from $k_{80...109}$, and find the matching $k_{0...77}$ in T_1. Next, based on the knowledge of $k_{0...79}$, compute the sub-key pair of k_{f_2} and k_{b_2}, and check whether the pair is in the product set of T_2 and T'_2. If so, do further brute-force testing on the candidate key.

The time complexity is

$$2^{78} \cdot \frac{39}{130} + 2^{78} \cdot \frac{59}{130} + 2^{48} \cdot \left(2^{32} \cdot \frac{16}{130} + 2^{32} \cdot \frac{16}{130} \right) + 2^{80-2} \approx 2^{79.45}.$$

The memory complexity is $2^{78} + 2^{78} + 2^{32} \approx 2^{79}$. The data complexity is $\lceil 80/48 \rceil = 2$ known plaintext-ciphertext pairs.

The detailed computation steps of the partial matching used in the second MITM part of the attack on KATAN48 are listed in Appendix A.

4.4.2 A 2D-MITM Attack on KATAN64

Our new attack on KATAN64 is similar as above. After searching all possible combinations of intermediate positions by programs, we find that it will allow us to attack more rounds

if the partial matching technique is performed in the first MITM part. The final number of the attacked rounds on KATAN64 is 112. The guessed state is $g = s_{65}$, the first MITM attack meets at s_{46}, and the second one meets at s_{73}. The attack steps are as follows.

1. Compute the 2 bits of $s_{46} = E_1(s_0)$ by using every possible $k_{f_1} = k_{0...77}$, and save all computation results in a table T_1 indexed by k_{f_1}.
2. Compute $s'_{73} = D_2(s_{112})$ by using every possible $k_{b_2} = k_{146...222}$, and compute $\kappa_{130...143}$ by treating $k_{144} = k_{145} = 0$. Save $k_{146...222}$'s into a table T'_2 indexed by s'_{72} and $\kappa_{130...143}$.
3. For each guess of $g = s_{65}$:
   (a) Compute the 2 bits of $s'_{46} = D_1(g)$ by using every possible $k_{b_1} = k_{114...129}$. Store all k_{b_1}'s in a table T'_1 indexed by the values of the 2 bits, and each entry is a sub-set containing certain k_{b_1}'s. After this step, T_1 and T'_1 together form a product set.
   (b) Compute $s_{73} = E_2(g)$ for each $k_{f_2} = k_{130...145}$, calculate $\kappa_{130...143}$ by subtracting k_{144} and k_{145} from $k_{130...143}$, and find the matching $k_{146...222}$ in T'_2. Next, based on the knowledge of $k_{144...223}$, compute the sub-key pair of k_{f_1} and k_{b_1}, and check whether the pair is in the product set of T_1 and T'_1. If so, do further brute-force testing on the candidate key.

The time complexity is

$$2^{78} \cdot \frac{46}{112} + 2^{78} \cdot \frac{39}{112} + 2^{48} \cdot \left(2^{32} \cdot \frac{19}{112} + 2^{32} \cdot \frac{8}{112} \right) + 2^{80-2} \approx 2^{79.45}.$$

The memory complexity is $2^{78} + 2^{78} + 2^{16} \approx 2^{79}$. The data complexity is $\lceil 80/64 \rceil = 2$ known plaintext-ciphertext pairs.

The steps of the partial matching for the first MITM part of the attack on KATAN64 are given in Appendix A.

4.5 Further Optimization Methods

There are still many techniques that may help us reduce the attacks' complexities and reach more rounds.

One way to reduce the time complexities is that when computing intermediate states for partial matching, we do not actually need to complete the calculations of partial encryptions/decryptions. Consider the detailed steps of the partial matching used in the 2D-MITM attack on KATAN32 (see Section 4.3.1). One of the two bits used for matching in s'_{87} has already been obtained after the decryption of the 106th round, and the other bit is computed in the 104th round. So we do not need to continue the partial decryptions after that. Moreover, the computations of these two bits depend on only parts of previous states, and thus we may also be able to save some time on the computations before the 104th round. But this technique will not push our attacks to more rounds, and might make the attack procedures very complicated to explain. In addition, in order to make our complexity estimations generous, this optimization method is not used in our attacks.

Another way to improve the attacks is to segment the ciphers' round functions into smaller steps. For example, the round functions of KATAN48 and KATAN64 update the internal states by two and three times, respectively, so we may divide them to two or three sub-functions. In addition, we can even separate operations of updating L_1 and L_2 to different sub-steps, which is applicable to any KATAN variant. By analyzing iterations of smaller steps or functions, we may further reduce time and memory complexities, or extend attacks to more rounds.

The paper [120] proposes an improved partial matching technique called *indirect partial matching*, in order to obtain more numbers of usable intermediate bits for matching. Originally, when computing a partial matching state, if the value of one bit s_i depends on the key bit k_j only known to its opposite phase, then s_i will be considered as *unknown*. Nonetheless, after adding this key bit k_j into computations, k_j may still remain as a linear variable in intermediate states after a few rounds. Thus, if we look for possible matches of $s_i \oplus k_j$ instead of s_i, this bit information can still be used for matching. This technique may help us extend our attacks to more rounds.

4.6 MD-MITM Attacks on KATAN with Less Rounds

Since the MD-MITM attacks in Sections 4.3 and 4.4 focus on increasing the maximum numbers of attacked rounds, their time complexities are close to exhaustive search's. It is still questionable that whether this new multidimensional approach can be practical enough to improve existing attacks on ciphers with less rounds. In this section, we demonstrate that how to apply MD-MITM attacks to reduced-round KATAN, and obtain less time complexities than the existing attacks in [5, 67].

4.6.1 A More Efficient Attack on 115-Round KATAN32

We show that how to attack KATAN32 with the exactly same number of rounds as in the paper [5], i.e., 115 rounds. For simplicity of description, we do not use any advanced optimization methods, such as partial matching.

This reduced-round attack is based on the one mentioned in Section 4.3.1, and our idea is to further reduce the time complexities by iterating less sub-key bits. The 115-round KATAN32 is segmented by s_{55}. The MITM part for the first sub-cipher meets at s_{35}, and the second MITM part meets at s_{79}. The attack procedure is similar to previous attacks, and can be generally described as follows.

1. The sub-key pairs, $k_{0...69}$ and $k_{70...109}$, in the first MITM phase form a product set.
2. The second MITM phase with $k_{110...157}$ and $k_{158...229}$ yields a consecutive 80-bit sub-key $k_{150...229}$. Then we recheck $k_{150...229}$ in the product set constructed in the first MITM phase.
3. Candidate keys are further examined by using additional pairs of plaintexts and ciphertexts.

The total time complexity of this new attack is

$$2^{70} \cdot \frac{35}{115} + 2^{72} \cdot \frac{36}{115} + 2^{32} \cdot \left(2^{40} \cdot \frac{20}{115} + 2^{48} \cdot \frac{24}{115} \right) + 2^{80-32} \approx 2^{77.75},$$

which is less than the time complexity of the attack in [5], i.e., 2^{79}. The memory complexity for this attack is $2^{70} + 2^{72} + 2^{40} \approx 2^{72.32}$.

4.6.2 A More Efficient Attack on 100-Round KATAN48

Similar to the attack on 115-round KATAN32, we can construct a simple 2D-MITM attack on 100-round KATAN48. The guessed state is s_{48}. The first MITM part meets at s_{35} and the second one meets at s_{64}. $k_{0...69}$ and $k_{70...95}$ in the first MITM phase can form a product set, and $k_{120...199}$ derived from the second MITM phase is checked again by using the product set.

The overall time complexity is

$$2^{70} \cdot \frac{35}{100} + 2^{72} \cdot \frac{36}{100} + 2^{48} \cdot \left(2^{26} \cdot \frac{13}{100} + 2^{32} \cdot \frac{16}{100} \right) + 2^{80-48} \approx 2^{77.37}.$$

Although this simple attack may be further optimized, it already has less time complexity than the attack on KATAN48 in [67]. The memory complexity of this new attack is $2^{70} + 2^{72} + 2^{26} \approx 2^{72.32}$, which is also less than the one in [67].

4.6.3 Discussions

A simple MD-MITM attack without any optimization will not get much advantage over the traditional MITM attacks on KATAN64 with 94 rounds (the maximum number of rounds attacked in [67]), since its block size, i.e., 64 bits, is close to the key size, 80 bits. It is feasible to get lower time complexity by adopting the partial matching as used on KATAN64 in Section 4.3.1. But as we already demonstrate the power of MD-MITM attacks on 115-round KATAN32 and 100-round KATAN48, and our primary goal is to extend attacks on KATAN to as many rounds as possible, we leave the potential work on 94-round KATAN64 to interested readers.

One subtle part in our attacks on KATAN, including the ones in Sections 4.3 and 4.4, is first deriving a consecutive sub-key for later use, such as $k_{151...230}$ used in the new attack on 115-round KATAN32. This is the most time-consuming step during the whole attack, and limits the lower bounds of time complexities for possible attacks. But since the computational cost of this step is low compared with a complete encryption/decryption of the cipher, the overall complexities of our attacks could be better than exhaustive search. However, we do not rule out the possibility of other ways to carry out MD-MITM attacks that may improve upon our methods.

The attacks on 115-round KATAN32 and 110-round KATAN48 proposed in this section demonstrate that this MD-MITM method not only can increase the numbers of possibly attacked rounds, but also could be used to improve the time and memory efficiencies of attacks on reduced-round ciphers. Compared with traditional MITM attacks that may need to iterate most of their attacking steps, MD-MITM approach can save computation consumption in certain phases, such as the first forward and last backward phases. This might also be one of the reasons why traditional MITM attacks cannot reach the same numbers of the attacked rounds of KATAN as MD-MITM attacks.

4.7 Summary

In this chapter, we have investigated a cryptanalysis framework called multidimensional meet-in-the-middle attack. This framework is applicable to lightweight ciphers with simple

key scheduling algorithms and block sizes smaller than master key sizes. Refined analysis and attacks have been presented on the block cipher family KATAN32/48/64 for demonstration. Our new attacks have reached more rounds than the existing attacks, and can also be more efficient than the existing ones when applied to KATAN with smaller numbers of rounds. Our new cryptanalysis results on KATAN are summarized in Table 4.3, where the notation KPs stands for *known plaintext-ciphertext pairs*.

Table 4.3: Comparisons of cryptanalysis results on reduced-round KATAN ciphers.

Algorithms	Rounds	Time Compl.	Memory Compl.	Data Compl.	Reference
KATAN32	115	2^{79}	Not Given	2^{32} KPs	[5]
	115	$2^{77.75}$	$2^{72.32}$	3 KPs	Section 4.6.1
	119	$2^{79.10}$	$2^{79.10}$	144 KPs	[66]
	175	$2^{79.30}$	$2^{79.58}$	3 KPs	Section 4.3.2
KATAN48	100	2^{78}	2^{78}	128 KPs	[67]
	100	$2^{77.37}$	$2^{73.32}$	2 KPs	Section 4.6.2
	105	$2^{79.10}$	$2^{79.10}$	144 KPs	[66]
	130	$2^{79.45}$	$2^{79.00}$	2 KPs	Section 4.4.1
KATAN64	94	$2^{77.68}$	$2^{77.68}$	116 KPs	[67]
	99	$2^{79.10}$	$2^{79.10}$	142 KPs	[66]
	112	$2^{79.45}$	$2^{79.00}$	2 KPs	Section 4.4.2

Chapter 5

Designing Password Hashing and Key Derivation Algorithms

Password-based authentication has been widely used in cloud services due to its simplicity and efficiency, as we have mentioned in Section 1.2.3. Our password hashing algorithms designs, PLECO and PLECTRON, are introduced in this chapter. Section 5.1 first gives certain background knowledge, before the designs of PLECO and PLECTRON are specified in Section 5.2. We discuss our design rationale and provide the security analysis of PLECO and PLECTRON in Section 5.3. Several extensions of the new hashing algorithms are proposed in Section 5.4, followed by the efficiency analysis in Section 5.5. We give a brief analysis of other password hashing designs in Section 5.6 and finally summarize this chapter in Section 5.7.

5.1 Preliminaries

This section first discusses some desired properties of a good password hashing algorithm, and then introduces the cryptographic primitives that PLECO and PLECTRON employ.

5.1.1 Desired Features of Password Hashing Algorithms

Password hashing is one of the most basic security considerations for setting up a password-based authentication system or deriving cryptographic keys from passwords, and there are several requirements that a good password hashing algorithm should fulfill:

- Similar as most cryptographic primitives, the password hashing algorithm should behave as a random function that ensures *one-wayness* and *collision resistance*, and is resistant to side-channel attacks as well as known cryptanalytic technologies such as time-memory tradeoff [41, 62], and differential/linear cryptanalysis [25, 83];
- Different from most cryptographic primitives, the password hashing algorithm should be *heavyweight* in computation and memory usage to slow down brute-force attacks to a certain degree and make large-scale attacks economically difficult. Note that the desired heavyweightness is expected to be roughly consistent for all platforms, no matter software or hardware;
- *Server-specific shortcut* is an optional but very attractive feature for a password hashing scheme. Once this feature is enabled and certain private information is known, legitimate servers or devices can obscure passwords by using less computation (*server-specific computational shortcut*) and/or less memory (*server-specific memory shortcut*).

Due to the uncommon and demanding features that are different from common cryptographic designs, the choices of well-studied password hashing algorithms are very limited.

5.1.2 Components of Pleco and Plectron

This section briefly describes several cryptographic primitives, which are the core components in the designs of Pleco and Plectron.

Provably One-Way Function Rabin_n

It is proven that the one-wayness of the Rabin public-key encryption scheme is as strong as the hard problem of integer factorization [102]. More theoretically, let us define

$$\mathsf{Rabin}_n(x) = x^2 \bmod n,$$

where x is a positive integer in the multiplicative group of integers modulo n. Then computing the square roots, i.e., inverting the function $\mathsf{Rabin}_n(x)$, is proven to be computationally equivalent to factorizing the integer n.

To obtain a hard-to-factor modulus n, one can utilize the same approach as generating moduli for the RSA algorithm, i.e., randomly generating two large prime numbers p and q,

and using their product $n = p \cdot q$ as a modulus. The other approach is to choose certain large composite numbers with unknown factorization, e.g., the Mersenne composite number used in Shamir's SQUASH construction [114]. More about publicly auditable moduli is discussed in Section 5.4.2.

Sponge-Based Hash Function Keccak

Keccak, which is designed by Bertoni *et al.* [23], is the winner of the SHA-3 cryptographic hash function competition held by NIST [44]. Keccak is based on a unique construction, namely *sponge construction*, which can *absorb* an arbitrary-length binary string as input, and then *squeeze* out a binary string of any required length as output.

In our password hashing designs, Keccak is adopted to:

- Fully mix password and salt strings;
- Expand short input strings to the large space of the Rabin encryption scheme;
- Alternately apply to intermediate states with the Rabin encryption scheme (or other public-key schemes) to gain more cryptanalytic strength;
- Process the final state to produce hash tags of required lengths.

If not specified, the default parameters of Keccak are used, i.e., $r = 1024$ and $c = 576$.

Sequential Memory-Hard Construction ROMix

The password-based key derivation function scrypt was proposed by Percival in order to thwart parallel brute-force attacks using GPUs, FPGAs or ASICs on passwords, and has been widely used by cryptocurrencies. One of core components of scrypt, namely ROMix, is proven to be *sequential memory-hard*, which implies [98]:

1. The construction ROMix asymptotically uses almost as many operations as memory locations; and
2. Parallel algorithms to compute ROMix cannot asymptotically achieve efficiency advantages than non-parallel ones.

Thus, for the brute-force password search using dedicated hardware with constrained memory, such as GPUs, FPGAs, and ASICs: 1) It would not be significantly faster than a single-core computer, as these dedicated hardwares usually have limited memory capacities; 2) Their strong power in parallel computations could not help in reducing the overall running time significantly.

We list ROMix in Algorithm 5.1 since it is highly relevant to our designs of Pleco and Plectron. In Algorithm 5.1, H is a cryptographic hash function, *bstr* is a binary string, *cost* is called the time or space cost parameter that must be larger than one, and Integerify is a bijective function that maps binary strings to integers.

Algorithm 5.1: ROMix(*bstr*, *cost*)

1: $x \leftarrow bstr$
2: **for** $i \leftarrow 0$ to $cost - 1$ **do**
3: $\quad v_i \leftarrow x$
4: $\quad x \leftarrow H(x)$
5: **end for**
6: **for** $i \leftarrow 0$ to $cost - 1$ **do**
7: $\quad j \leftarrow \mathsf{Integerify}(x) \bmod cost$
8: $\quad x \leftarrow H(x \oplus v_j)$
9: **end for**
10: **return** x

5.2 Designs of Pleco and Plectron

The design rationale of Pleco and Plectron is to inherit the existing structure of scrypt that is proved to be sequential memory-hard, and to improve its inner components for providing better security and asymmetry in computation as desired.

The following notations are used in this section, where the notations $||$, 0^t and $\mathsf{len}(\cdot)$ have been introduced in Section 2.1, but for ease of reference we describe them again here. Please also note that $\mathsf{int}(\cdot)$ and $\mathsf{str}_b(\cdot)$ are different from the ones in Section 2.1 regarding endianness, i.e., the positions for least/most significant bit.

- $||$ concatenates two binary strings;

- $\mathsf{int}(s)$ converts a binary string s into a non-negative integer, where the little-endian convention is used, i.e., the left-most (lowest address) bit is the least significant bit of the integer[1];
- $\mathsf{str}_b(x)$ converts a non-negative integer x back to a binary string by using the same bit ordering convention as $\mathsf{int}(\cdot)$, and may append zeros to the string in order to achieve a total length of b bits;
- 0^t denotes a t-bit all-zero binary string, i.e., $0^t = \mathsf{str}_t(0)$ for $t > 0$, and 0^0 means an empty string;
- $\mathsf{len}(s)$ denotes the bit-length of the binary string s;
- $\mathsf{size}(x)$ denotes the number of bits in the shortest binary representation of the given positive integer x, e.g., $\mathsf{size}(256) = 9$ and $\mathsf{size}(255) = 8$;
- Keccak_b denotes a Keccak instance that produces exactly b bits as output.

Given a positive integer n, we define a new hash function

$$\mathcal{H}_n(x) = \mathsf{str}_N(\mathsf{Rabin}_n(1 + \mathsf{int}(\mathsf{Keccak}_{N-1}(x)))),$$

where $N = \mathsf{size}(n)$.

To be secure, N should be at least 1024, or preferably larger than 3072. According to [15], a 1024-bit modulus would aim at a security level of about 80 bits, which means that on average it takes $2^{80-1} = 2^{79}$ time units for attacks.

As we have mentioned in Section 5.1.2, the modulus n can be obtained using the same approach for generating the RSA modulus $n = p \cdot q$, or chosen from a public composite number with unknown factorization as proposed in the design of SQUASH [114].

Our new password hashing algorithm Pleco is defined by Algorithm 5.2, which takes as input

- a positive integer n as the modulus,
- a 128-bit binary string *salt* as a unique or randomly generated salt,

[1]For software implementations, we recommend using the following convention: The 8 least significant bits are stored in the byte with the lowest address, and within a byte the least significant bit is the coefficient of 2^0. This follows the internal implementation convention of Keccak [24].

Algorithm 5.2: PLECO(n, *salt*, *pass*, *tcost*, *mcost*)

1: $L \leftarrow 8 \cdot \lceil \mathsf{size}(n)/8 \rceil - \mathsf{size}(n)$
2: $x \leftarrow salt||\mathsf{str}_{16}(\mathsf{len}(pass))||pass||0^{1024-\mathsf{len}(pass)}$
3: $ctr \leftarrow 0$
4: $x \leftarrow \mathcal{H}_n(\mathsf{str}_{128}(ctr)||x)$
5: **for** $i \leftarrow 0$ to $tcost - 1$ **do**
6: **for** $j \leftarrow 0$ to $mcost - 1$ **do**
7: $v_j \leftarrow x$
8: $ctr \leftarrow ctr + 1$
9: $x \leftarrow \mathcal{H}_n(\mathsf{str}_{128}(ctr)||x)$
10: **end for**
11: **for** $j \leftarrow 0$ to $mcost - 1$ **do**
12: $k \leftarrow \mathsf{int}(x) \bmod mcost$
13: $ctr \leftarrow ctr + 1$
14: $x \leftarrow \mathcal{H}_n(\mathsf{str}_{128}(ctr)||x||0^L||v_k)$
15: **end for**
16: $ctr \leftarrow ctr + 1$
17: $x \leftarrow \mathcal{H}_n(\mathsf{str}_{128}(ctr)||x)$
18: **end for**
19: **return** x

- a variable-length (≤ 128 bytes) binary string *pass* as a user password,
- a positive integer *tcost* as the time cost parameter, and
- a positive integer *mcost* as the memory cost parameter.

Lines 6-15 of Algorithm 5.2 are essentially the same as ROMix, except that:

- Instead of XORing v_k with x as the design of ROMix, we concatenate them and input into $\mathcal{H}_n$;
- An incremental counter *ctr* is always prepended to the intermediate variable x in each step.

Sections 5.3.2 and 5.3.4 will give the detailed reasons why we introduce these changes.

PLECO will produce a $\mathsf{size}(n)$-bit hash tag, but sometimes applications need to flexibly choose tag sizes, e.g., generating cryptographic keys from passphrases entered by users.

We recommend applying Keccak to the output of PLECO again to produce tags of required lengths. We name this modified algorithm as PLECTRON and specifies its design in Algorithm 5.3, where

- *hsize* denotes the desired bit-length of the hash tag.

Algorithm 5.3: PLECTRON(n, *salt*, *pass*, *tcost*, *mcost*, *hsize*)

1: $t \leftarrow$ PLECO(n, *salt*, *pass*, *tcost*, *mcost*)
2: **return** $\mathsf{Keccak}_{hsize}(t)$

5.3 Security Analysis

The designs of PLECO and PLECTRON combine public-key and symmetric-key algorithms and alter the operation sequence to make cryptanalysis harder. This is analogous to the designs of ARX ciphers and the block cipher IDEA, where mixed operations are used. In what follows, we discuss the security properties of PLECO and PLECTRON in detail.

5.3.1 One-Wayness

One of the most important security goals of designing a password hashing scheme is one-wayness, i.e., attackers should not be able to devise any methods faster than the exhaustive search for inverting the hashing algorithm in order to obtain the original passwords.

In our designs, the cryptographic hash function Keccak and the provably one-way function Rabin_n are applied to the intermediate state x alternatively. To the best of our knowledge, no weaknesses have been reported when combining these two algorithms. In order to invert $\mathcal{H}_n$, the attackers may have to analyze Keccak and Rabin_n separately. On one hand, even if the one-wayness of Keccak is completely broken, say replacing Keccak by an identity function, the one-wayness of $\mathcal{H}$ is still guaranteed by Rabin_n, i.e., the hardness of integer factorization. On the other hand, if any weakness of iterating Keccak is found, the weakness is highly likely to be covered up by the computations of Rabin_n.

More formally, we give the following definition.

Definition 5.1. *For a given function f and a pre-specified set Y containing certain outputs of f, we define the advantage of an adversary $\mathcal{A}$ obtaining preimages of the elements in Y (i.e., inverting f) as*

$$\mathbf{Adv}_{f}^{\mathrm{Pre}(Y)}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr[y \xleftarrow{\$} Y, x \leftarrow \mathcal{A}^{f,y} : f(x) = y],$$

where $y \xleftarrow{\$} Y$ means randomly assigning one element of Y to y.

Then we can show the preimage security of $\mathcal{H}_n$ is guaranteed by Rabin_n, as described in the following lemma.

Lemma 5.1 (One-Wayness of $\mathcal{H}_n$)**.** *For any adversary $\mathcal{A}$, we have*

$$\mathbf{Adv}_{\mathcal{H}_n}^{\mathrm{Pre}(S)}(\mathcal{A}) \leq \mathbf{Adv}_{\mathsf{Rabin}_n}^{\mathrm{Pre}(S)}(\mathcal{A}),$$

where S is a set consisting of certain outputs of $\mathcal{H}_n$.

Proof. Assume that n is an N-bit modulus. Once a preimage of $\mathcal{H}_n$ is found, e.g., $y = \mathcal{H}_n(x)$, we let $x' = 1 + \mathsf{int}(\mathsf{Keccak}_{N-1}(x))$ and $y' = \mathsf{int}(y)$, and then x' is a preimage of y' of Rabin_n. □

For a reasonably large set S, computing preimages of Rabin_n regarding S is still as hard as factoring the integer n, since the factorization will be known after obtaining a constant number of preimages on average. For example, for RSA-like moduli, the expected number of preimages required is 2 [88].

Please note that Lemma 5.1 presents a simplified bound only for the case that n is not factored by adversaries. If the factorization of n is known to adversaries, the one-wayness of $\mathcal{H}$ is still guaranteed by Keccak.

Based on Lemma 5.1, we can investigate the one-wayness of the whole design of Pleco.

Theorem 5.1 (One-Wayness of Pleco)**.** *If* Pleco *and $\mathcal{H}_n$ use a same modulus n, then we have*

$$\mathbf{Adv}_{\textsc{Pleco}}^{\mathrm{Pre}(S)}(\mathcal{A}) \leq \mathbf{Adv}_{\mathsf{Rabin}_n}^{\mathrm{Pre}(S)}(\mathcal{A}),$$

where S is a set containing all possible outputs of Pleco.

Proof. Assume that a preimage of PLECO is found, then the preimage of the last $\mathcal{H}_h$ can be obtained if we recompute the steps of PLECO before the last $\mathcal{H}_n$. Thus, we have

$$\mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathrm{PLECO}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathcal{H}_n}(\mathcal{A}),$$

which implies

$$\mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathrm{PLECO}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathsf{Rabin}_n}(\mathcal{A}),$$

due to Lemma 5.1. □

For the preimage in the above theorem, we do not differentiate the two cases: 1) a preimage containing both *salt* and *pass*, or 2) a preimage including only *pass* for a pre-specified *salt*. For the second case, the first Keccak_{N-1} in PLECO can be seen as a specialized Keccak instance, as the design of Keccak supports simply prepending a message with a key to construct a message authentication code (MAC) algorithm [23]. Therefore, an adversary's advantage of recovering *pass* still satisfies the bound in Theorem 5.1, even if *salt* is public or known to adversaries.

Next, let us consider the one-wayness of PLECTRON. If we assume that, in order to invert PLECTRON, any adversary has to first invert Keccak_{hsize} and then invert PLECO, we can simply get a bound like

$$\begin{aligned}\mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathrm{PLECTRON}}(\mathcal{A}) &\leq \mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathsf{Keccak}_{hsize}}(\mathcal{A}) \cdot \mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathrm{PLECO}}(\mathcal{A}) \\ &\leq \mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathsf{Keccak}_{hsize}}(\mathcal{A}) \cdot \mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathsf{Rabin}_n}(\mathcal{A}).\end{aligned}$$

However, it cannot be guaranteed that adversaries will always try to obtain the intermediate value between PLECO and Keccak_{hszie}. Consider the case where *hsize* is very small, say two bits. After trying random passwords for four times, on average there will be one password producing the 2-bit pre-specified hash tag. Therefore, in theory, we can only give the following theorem on the one-wayness of PLECTRON.

Theorem 5.2 (One-Wayness of PLECTRON)**.** *If $\mathcal{H}_n$ and* PLECTRON *use a same modulus n, then we have*

$$\mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathrm{PLECTRON}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathsf{Keccak}_{hsize}}(\mathcal{A}),$$

where S is a set containing all possible outputs of PLECTRON*.*

Proof. Once a primage of PLECTRON is found, e.g.,

$$y = \mathrm{PLECTRON}(n, s, p, tc, mc, hsize),$$

we compute

$$x = \mathrm{PLECO}(n, s, p, tc, mc).$$

Then x is a preimage of y of Keccak_{hsize}. □

As a cryptographic hash function, Keccak is designed to be preimage-resistant, which means that for essentially all outputs, finding any input hashing to a pre-specified output should be computationally infeasible [88, 105].

5.3.2 Collision Resistance

Collision and second-preimage resistances are also desirable when designing a password hashing scheme. In this context, an occurrence of collision may result in two passwords being hashed to the same tag, whereas a second-preimage implies that given a password $pass_1$, one may find the second one $pass_2$ producing the same tag. It is easy to see that if there exists an algorithm for constructing second-preimages, then it can also be used to generate collisions, so the collision resistance implies the second-preimage resistance.

It is easy to see that once a collision of Keccak_{N-1} is found, then it will result in a collision of $\mathcal{H}_n$. Furthermore, if the outputs of Keccak contain two roots of Rabin_n, then it will also produce a collision of $\mathcal{H}_n$. Therefore, the collision resistance of $\mathcal{H}_n$ is bounded by properties of Rabin_n and Keccak together.

Formally, we give the following security definition.

Definition 5.2. *For a given function f, we define the advantage of an adversary $\mathcal{A}$ to find a collision of f as*

$$\mathbf{Adv}_f^{\mathrm{Coll}}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr[x_1, x_2 \leftarrow \mathcal{A}^f : f(x_1) = f(x_2)].$$

To better analyze the collision resistance of $\mathcal{H}_n$, we give the following the definitions.

Definition 5.3. *For a given function f, we define the advantage of an adversary $\mathcal{A}$ to obtain an output difference d as*

$$\mathbf{Adv}_f^{\mathrm{Diff}(d)}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr[x_1, x_2 \leftarrow \mathcal{A}^{f,d} : f(x_1) = d - f(x_2)].$$

Definition 5.4. *For a given positive composite integer m, we define the advantage of an adversary $\mathcal{A}$ to obtain a non-trivial factor of m as*

$$\mathbf{Adv}_m^{\mathrm{Fact}}(\mathcal{A}) \stackrel{\mathrm{def}}{=} \Pr[x \leftarrow \mathcal{A}^m : x | m, 1 < x < m].$$

This advantage $\mathbf{Adv}_f^{\mathrm{Diff}(d)}(\mathcal{A})$ should be negligible for any secure cryptographic hash function f, since these hash functions are designed to be indistinguishable from pseudo-random functions.

Then we have the following lemma about collisions of $\mathcal{H}_n$.

Lemma 5.2 (Collision Resistance of $\mathcal{H}_n$)**.** *For any adversary $\mathcal{A}$, we have*

$$\mathbf{Adv}^{\mathrm{Coll}}_{\mathcal{H}_n}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Coll}}_{\mathsf{Keccak}_{N-1}}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{Diff}(n)}_{\mathsf{Keccak}_{N-1}}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{Fact}}_{n}(\mathcal{A}),$$

where $N = \mathsf{size}(n)$.

Proof. Suppose a colliding pair x_1 and x_2 of $\mathcal{H}_n$ are found, i.e., $\mathcal{H}_n(x_1) = \mathcal{H}_n(x_2)$. Let $r_1 = \mathsf{Keccak}_{N-1}(x_1)$ and $r_2 = \mathsf{Keccak}_{N-1}(x_2)$. Then we have three cases:

- If $r_1 = r_2$, then a collision of Keccak_{N-1} is found;
- If $r_1 \neq r_2$, let $s_1 = 1 + \mathsf{int}(r_1)$ and $s_2 = 1 + \mathsf{int}(r_2)$, and then:
  - If $s_1 = n - s_2$, then a pair producing the output difference n of Keccak_{N-1} is found;
  - If $s_1 \neq n - s_2$, then $\gcd(s_1 - s_2, n)$ is a non-trivial factor of n.

Therefore, the lemma holds. □

As the previous discussion about Lemma 5.1, Lemma 5.2 also gives a simplified bound on collisions that satisfies our purpose of showing $\mathcal{H}_n$ to be secure. Even if n is factored, it should still be hard to construct collisions of the whole $\mathcal{H}_n$, since adversaries need to control outputs of Keccak_{N-1} to be among roots corresponding to a same squaring value. For example, for RSA-like moduli, there are only four roots mapping to one output.

Based on Lemma 5.2, we give the following theorem to characterize adversaries' collision advantage on PLECO.

Theorem 5.3 (Collision Resistance of PLECO)**.** *If the cost parameters, mcost and tcost, of* PLECO *keep unchanged, and $\mathcal{H}_n$ and* PLECO *use a same N-bit modulus n, then we have*

$$\mathbf{Adv}^{\mathrm{Coll}}_{\mathrm{PLECO}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Coll}}_{\mathsf{Keccak}_{N-1}}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{Diff}(n)}_{\mathsf{Keccak}_{N-1}}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{Fact}}_{n}(\mathcal{A}).$$

Proof. Once a collision of PLECO is found, then there must exist a collision of the internal hash function $\mathcal{H}_n$. Thus, we have

$$\mathbf{Adv}^{\mathrm{Coll}}_{\mathrm{PLECO}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Coll}}_{\mathcal{H}_n}(\mathcal{A}).$$

Therefore, the theorem holds. □

Please note that if we use the original design of ROMix, i.e., XORing x and v_k instead of concatenating them together as input, the bound for collisions will be much more difficult to be discovered and proven, because different intermediate values x_1 and x_2 may still yield an identical input to the internal hash function $\mathcal{H}_n$, e.g., $x_1 \oplus v_{k_1} = x_2 \oplus v_{k_2}$. However, in the current design of PLECO, different x_1 and x_2 will never generate identical inputs to $\mathcal{H}_n$.

For PLECTRON, we have the following theorem.

Theorem 5.4 (Collision Resistance of PLECTRON)**.** *If the cost parameters and output hash length, mcost, tcost and hsize, of* PLECTRON *keep unchanged, and $\mathcal{H}_n$ and* PLECTRON *use a same N-bit modulus n, then we have*

$$\begin{aligned}\mathbf{Adv}^{\mathrm{Coll}}_{\textsc{Plectron}_n}(\mathcal{A}) &\leq \mathbf{Adv}^{\mathrm{Coll}}_{\mathsf{Keccak}_{N-1}}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{Diff}(n)}_{\mathsf{Keccak}_{N-1}}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{Fact}}_{n}(\mathcal{A}) \\ &+ \mathbf{Adv}^{\mathrm{Coll}}_{\mathsf{Keccak}_{hsize}}(\mathcal{A}).\end{aligned}$$

Proof. If a collision of PLECTRON is found, e.g.,

$$\begin{aligned}&\textsc{Plectron}(n, s_1, p_1, tc, mc, hsize) \\ =\ &\textsc{Plectron}(n, s_2, p_2, tc, mc, hsize),\end{aligned}$$

then we let

$$\begin{cases} t_1 = \textsc{Pleco}(n, s_1, p_1, tc, mc) \\ t_2 = \textsc{Pleco}(n, s_2, p_2, tc, mc) \end{cases}.$$

We have the following two cases:

- If $t_1 = t_2$, then a collision of PLECO is found.
- If $t_1 \neq t_2$, then a collision of Keccak_{hsize} is found.

Therefore, we have

$$\mathbf{Adv}^{\mathrm{Coll}}_{\textsc{Plectron}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Coll}}_{\textsc{Pleco}}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{Coll}}_{\mathsf{Keccak}_{hsize}}(\mathcal{A}).$$

Thus, the theorem holds. □

5.3.3 Thwarting Parallel Brute-Force Attacks

Although the designs of Pleco and Plectron may be secure for random inputs in theory, users' passwords are usually weak and easily crackable by using parallel search based on dedicated or custom-designed hardware, such as GPUs, FPGAs, and ASICs. Thus password hashing designs should thwart such attacks as much as possible.

The hardware such as GPUs, FPGAs, and ASICs can feature thousands of cores for parallel computation, but in return each core possesses very restrained memory space. By using the structure of ROMix, the internal construction of Pleco (Lines 6-15 in Algorithm 5.2) inherits scrypt's security property of being sequential memory-hard. Pleco and Plectron also provide a tunable memory parameter *mcost* to increase their memory cost as desired. Although the design of Pleco is slightly different from ROMix, the security proofs of ROMix can be easily transferred to here, since in the original proofs the internal hash function is treated as a *Random Oracle*.

Please note that in scrypt, a structure called BlockMix is used to build an internal hash function with wide input/output from a small function Salsa20 core [22]. However, BlockMix is not necessary for Pleco since the input/output lengths of $\mathcal{H}_n$ are relatively large. As a side benefit of omitting BlockMix, our scheme is simpler and easier for analysis, when compared with scrypt.

5.3.4 Preventing Self-Similarity Attacks

An incremental counter *ctr* is prepended to the intermediate state of Pleco before each invocation of $\mathcal{H}_n$, which enables us to protect Pleco and Plectron from certain potential self-similarity attacks, such as fixed points or iterative patterns of $\mathcal{H}_n$. The similar technique is used in many other cryptographic designs, such as Keccak, PRESENT [28] and PRINCE [31].

5.4 Other Extensions

In this section, we propose a number of variants and potential use cases of Pleco and Plectron.

5.4.1 Discrete-Logarithm-Based Hash Function

Gibson has proved that if factoring n is hard, the following discrete-logarithm-based hash function

$$\mathcal{G}_n(x) = g^x \bmod n$$

is one-way and collision-free [53][2], where x is a positive integer and g is a generator of the multiplicative group of integers modulo n. The security of this hash function is guaranteed by the hardness of integer factorization, since a collision will lead to the factorization of n.

We define a new hash function

$$\mathcal{GH}_n(x) = \mathsf{str}_N(\mathcal{G}_n(1 + \mathsf{int}(\mathsf{Keccak}_N(x)))),$$

for a given positive integer n, where $N = \mathsf{size}(n)$. If $\mathcal{H}_n(x)$ in PLECO is replaced by $\mathcal{GH}_n(x)$, the security, especially the collision resistance, of PLECO and PLECTRON would be further enhanced.

Lemma 5.3. *For any adversary $\mathcal{A}$, we have*

$$\mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathcal{GH}_n}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Fact}}_{n}(\mathcal{A}),$$

and

$$\mathbf{Adv}^{\mathrm{Coll}}_{\mathcal{GH}_n}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Coll}}_{\mathsf{Keccak}_N}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{Coll}}_{\mathcal{G}_n}(\mathcal{A}),$$

where S is a set consisting of certain outputs of $\mathcal{GH}_n$.

Proof. The proofs are similar to the ones for Lemmas 5.1 and 5.2, so they are omitted here. □

It is easy to see that $\mathbf{Adv}^{\mathrm{Coll}}_{\mathcal{GH}_n}(\mathcal{A})$ is smaller than $\mathbf{Adv}^{\mathrm{Coll}}_{\mathcal{H}_n}(\mathcal{A})$, because $\mathbf{Adv}^{\mathrm{Coll}}_{\mathcal{G}_n}(\mathcal{A})$ here should be equivalent to the term $\mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathsf{Rabin}_n}(\mathcal{A})$ in Lemma 5.1, but $\mathbf{Adv}^{\mathrm{Coll}}_{\mathcal{H}_n}(\mathcal{A})$ has an extra $\mathbf{Adv}^{\mathrm{Diff}(n)}_{\mathsf{Keccak}_{N-1}}(\mathcal{A})$ (see Lemma 5.2).

We have the following theorem about using $\mathcal{GH}_n$ in PLECO.

[2]It is claimed in [113] that this hash function was proposed by Shamir, and a simple proof was given by Rivest.

Theorem 5.5. *Suppose the cost parameters, mcost and tcost, of* PLECO *keep unchanged, and $\mathcal{GH}_n$ and* PLECO *use a same N-bit modulus n. If replacing $\mathcal{H}_n$ by $\mathcal{GH}_n$ in* PLECO*, namely* PLECO$'$*, we will have*

$$\mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathrm{PLECO}'}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Pre}(S)}_{\mathcal{G}_n}(\mathcal{A}),$$

and

$$\mathbf{Adv}^{\mathrm{Coll}}_{\mathrm{PLECO}'}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{Coll}}_{\mathsf{Keccak}_N}(\mathcal{A}) + \mathbf{Adv}^{\mathrm{Coll}}_{\mathcal{G}_n}(\mathcal{A}),$$

where S is a set containing all possible outputs of PLECO$'$*.*

Proof. As the proofs for Theorems 5.1 and 5.3, the one-wayness and collision resistance of PLECO$'$ are guaranteed by the security properties of $\mathcal{GH}_n$. Thus, the theorem holds. □

For PLECTRON, if $\mathcal{H}_n$ is replaced by $\mathcal{GH}_n$, its one-wayness (Theorem 5.2) does not change, but the bound for its collisions will be improved as PLECO.

Another benefit of using $\mathcal{GH}_n$ instead of $\mathcal{H}_n$ is that $\mathcal{G}_n$ is proven to be secure for any positive integer as input. Compared with Rabin_n, whose security properties only consider the inputs within the multiplicative group of integers modulo n, $\mathcal{G}_n$ allows much more flexibility when we adopt it to design security schemes. For example, in the design of $\mathcal{GH}_n$, the output length of Keccak may be equal to or larger than $\mathsf{size}(n)$; while in $\mathcal{H}_n$, Keccak is set to generate less than $\mathsf{size}(n)$ bits.

Although the discrete-logarithm-based hash function $\mathcal{G}_n$ is more secure and flexible, it is much less efficient than Rabin_n due to the slow modular exponentiation computations.

5.4.2 Using Publicly Auditable Modulus

As observed by Shamir in [114], the Rabin scheme cannot be efficiently inverted for any modulus n with unknown factorization. As a result, large composite Mersenne numbers of the form $n = 2^k - 1$ with unknown factorization can be used as the modulus, which enables efficient software implementation of PLECO and PLECTRON (see Section 5.5.2 for performance comparison). A table summarizing the factorization of Mersenne numbers of the form $M_k = 2^k - 1$ is maintained by Leyland [79]. Certain interesting Mersenne numbers that might be used as the moduli in PLECO and PLECTRON for different security levels[3] are $2^{1277} - 1$, $2^{2137} - 1$, and $2^{3049} - 1$.

[3]Shamir has chosen a slightly longer Mersenne number as the modulus for the implementation of SQUASH, because he expects that Mersenne numbers are easier to factor than general numbers. For more details, please refer to [114]. Here we simply choose $2^{1277} - 1$ since it has been used in [114], and $2^{2137} - 1$ and $2^{3049} - 1$ as their bit-lengths are close to 2048 and 3072, respectively.

Furthermore, RSA-like moduli might not be suitable if PLECO or PLECTRON are used in cryptocurrencies for proofs of work, because RSA-like moduli must be generated by someone. With the private knowledge of the factors of n, one may compute the hash functions more efficiently than others (which is discussed in Section 5.5.3). By using public composite numbers with unknown factors, we can eliminate potential trapdoors in cryptocurrency systems.

One potential method to publicly generate moduli with unknown factorization is constructing the numbers called RSA-UFOs proposed by Sander in [109]. But RSA-UFOs may be too large for practical applications, e.g., in order to be used as a 1024-bit modulus it may require more than 40000 bits. Moreover, RSA-UFOs will inevitably contain many known factors if they are generated by the method in [109]. Therefore, the performance and security of RSA-UFOs still need more investigations.

5.4.3 Transforming Existing Hash Tags to Larger Cost Settings

For PLECO, its final output can re-enter the algorithm from Line 6 (Algorithm 5.2), which is equivalent to increasing the time cost parameter *tcost* by one. During the additional computations, we can also choose a larger memory parameter *mcost*. Under such circumstance, hash tags can be updated according to new cost settings without the knowledge of original passwords.

5.4.4 Variants with More Efficient Software Implementations

In order to be easily and efficiently implemented in software, it is better for the modulus n to have a size that is a multiple of computer word sizes. But under certain circumstances, we cannot choose the size of n freely, e.g., using Mersenne composite numbers as moduli, so we may need to make small changes to the original algorithms of $\mathcal{H}_n$ and PLECO to achieve a better efficiency.

Let

$$\begin{cases} UB = w \cdot \lceil N/w \rceil \\ LB = w \cdot (\lceil N/w \rceil - 1) \end{cases},$$

where w is the desired word size and N is the size of the modulus n. Then we define the following modified version of $\mathcal{H}_n$.

$$\mathcal{RH}_n(x) = \mathsf{str}_{UB}(\mathsf{Rabin}_n(1 + \mathsf{int}(\mathsf{Keccak}_{LB}(x))))$$

By replacing $\mathcal{H}_n$ by $\mathcal{RH}_n$, the software performance may be improved, because operations are applied to a multiple of words. But if the inputs into Rabin_n are so small that their squaring results do not need to be modulo n, then adversaries can easily compute the original inputs. The smaller LB is, the higher the probability of Rabin_n getting such inputs will be. Therefore, LB should not be too small.

To unify lengths of internal variables, we may simply substitute LB with UB, and get the following hash function.

$$\mathcal{RH}'_n(x) = \mathsf{str}_{UB}(\mathsf{Rabin}_n(1 + \mathsf{int}(\mathsf{Keccak}_{UB}(x))))$$

The collision probability of $\mathcal{RH}'_n$ will be higher, as there are inputs larger than n that cause collisions, e.g., $\mathsf{Rabin}_n(x + n) = \mathsf{Rabin}_n(x)$. However, the overall security of the password hashing scheme may still be acceptable, since it will be difficult to construct inputs with such additional differences through Keccak.

It is also possible to remove the plus-one operation in $\mathcal{H}_n$, i.e., defining the following function to replace $\mathcal{H}_n$.

$$\mathcal{SH}_n(x) = \mathsf{str}_N(\mathsf{Rabin}_n(\mathsf{int}(\mathsf{Keccak}_{N-1}(x))))$$

There is a negligible chance that $\mathsf{Keccak}_{N-1}(x)$ outputs zero, and the result of $\mathcal{SH}_n$ will be an all-zero string. If we treat Keccak as a pseudorandom function, this probability will be $1/2^{N-1}$. Even if this incident happens, it will likely disappear when $\mathcal{SH}_n$ is iterated for multiple times with an incremental counter. Henceforth, the security level of the entire design of Pleco should not be influenced.

5.5 Performance Analysis

In this section, we discuss the time and memory costs of Pleco and Plectron.

5.5.1 Tunable Time and Memory Costs

The designs of Pleco and Plectron provide two parameters, *tcost* and *mcost*, for applications to tune their time and memory consumptions.

The parameter *mcost* adjusts the amount of memory that needs to be present during the computations of Pleco and Plectron. The memory usage is expected to be around

$$\mathsf{size}(n) \cdot mcost$$

bits. Due to the sequential memory-hard property inherited from ROMix, without having such amount of memory, the computation time of Pleco and Plectron will increase significantly.

The parameter *tcost* has limited ability to adjust the time usage of Pleco and Plectron, since the total time cost also relies on the memory usage. To complete a full computation of Pleco, it requires

$$2 \cdot mcost \cdot tcost + tcost + 1$$

invocations of $\mathcal{H}_n$. Plectron needs one more invocation of Keccak than Pleco.

Although our work only aims to provide flexible solutions that can be tuned by users or developers for different applications, we would like to discuss a little bit about how to choose cost parameters properly in practice. At the time of writing, the graphic cards (GPUs) on the market have up to thousands of cores and several gigabytes of memory, so each core may have couples of megabytes of memory on average. FPGA or ASIC based circuits usually have less memory per core than GPUs. Therefore, in order to effectively thwart capital-rich attackers for building large-scale searching circuits, the memory usage of password-hashing or proof-of-work algorithms should require tens of megabytes of memory at minimum, e.g., $tcost \geq 2^{16}$ for Pleco and Plectron. Consider the design of Litecoin, in which scrypt is configured to consume only 128 kilobytes of memory. In our opinion, 128 kilobytes are too conservative, and thus cannot fully remove the advantages of attackers equipped with dedicated hardware.

For the choices of the time cost parameter *tcost*, it should be fine to stay with the minimum numbers, say 1 or 2, unless certain memory-constrained application scenarios want more control on the computation time.

5.5.2 Efficiency of Software Implementations

Pleco and Plectron are built upon well-established cryptographic primitives, and their implementations have been studied for years. The modular squaring operation is the basis for efficient implementations of RSA encryption/signature widely used in TLS/SSL, and Keccak is designed to be efficient in both software and hardware.

We have tested our initial implementations of Pleco and Plectron on a 2.6 GHz Intel Core i7 processor for 80-, 112-, and 128-bit security levels. For each security level, an RSA-like modulus $n = p \cdot q$ as well as a Mersenne number with a similar bit-length (see Table 5.1) are chosen as the moduli in Pleco and Plectron, in order for performance

comparisons (see Table 5.2). We set $mcost = 2^{16}$ when profiling the software performance, which means the programs will consume $2^{16}\mathsf{size}(n)$-bit memory, i.e., around 17 MB for Pleco/Plectron using the modulus $2^{2137} - 1$. We have also tested scrypt on the same machine, using the configuration $(N = 2^{14}, r = 8, p = 1)$ that yields a similar memory usage as Pleco/Plectron with 2048-bit moduli, and it takes 35 ms to compute scrypt.

Table 5.1: Modulus choices for different security strengths.

Security Strength	Size of RSA-Like Modulus (in bits)	Mersenne Number
80-bit	1024	$2^{1277} - 1$
112-bit	2048	$2^{2137} - 1$
128-bit	3072	$2^{3049} - 1$

Table 5.2: Software performance of Pleco/Plectron with $tcost = 1$ and $mcost = 2^{16}$.

Modulus Size (in bits)	Pleco		Plectron	
	RSA-like	Mersenne	RSA-like	Mersenne
1024 / 1277	0.684 s	0.538 s	0.686 s	0.540 s
2048 / 2137	2.215 s	1.185 s	2.235 s	1.203 s
3072 / 3049	4.355 s	2.135 s	4.358 s	2.146 s

The performance data in Table 5.2 shows that as the modulus size grows, the running time of the algorithms Pleco/Plectron will increase (along with the memory usage), and computation of the Rabin_n part will gradually dominate the running time. Moreover, using Mersenne numbers as moduli will yield more efficient computations than choosing RSA-like ones.

Although slowness is somehow desirable in password hashing designs for thwarting large-scale password searching, we should consistently improve the time efficiency of the implementations of Pleco and Plectron. For example, by reducing the computation time of $\mathcal{H}_n$, we will have more flexibility for the time parameter *tcost*. Moreover, attackers are always trying to speed up their searching methods, so there is no reason why legitimate users or servers should stick to under-optimized implementations.

5.5.3 Shortcut with Private Information

It would be very attractive if password hashing algorithms could support private parameters or keys to speed up hashing computations. For example, legitimate servers with certain private information may compute or verify hash tags faster than the attackers who have obtained only salts and hash tags. In this way, the servers will save time and hardware costs without risking too much about the overall security.

Due to the nature of the modular exponentiation operation, if we know the factorization of the modulus n, e.g., knowing p and q for $n = pq$, the computation can be finished with less time, by using the Chinese Remainder Theorem (CRT). Such performance gain might not be obvious for Rabin_n, as its operations are simple. But if the discrete-logarithm-based hash function, $\mathcal{G}_n(x) = g^x \bmod n$, is used in $\mathcal{H}_n$, the computation will be greatly accelerated if the factors of n are known. But p and q should be kept securely as always, e.g., being encrypted or stored in a hardware security module.

Note that even if p and q are leaked to attackers, the overall security of PLECO and PLECTRON still has Keccak as a "fail-safe". With the private information, attackers can compute PLECO or PLECTRON as efficient as legitimate servers, but the brute-force search for the original passwords may still be a must.

5.6 Comparisons with Other Password Hashing Algorithms

In this section, we analyze several other hashing designs and compare them with PLECO and PLECTRON. Especially, many new designs have been proposed recently, since an open password hashing competition (PHC) is currently ongoing [4]. We only include couples of designs from the competition submissions to be discussed here, which may be the most typical ones or relevant to ours.

5.6.1 scrypt

As we have mentioned in Section 5.1.2, scrypt presents the idea along with the first concrete design of sequential memory-hard algorithms. The internal structure of our designs PLECO and PLECTRON are based on ROMix of scrypt.

However, the overall design of scrypt is complicated. It uses BlockMix and the Salsa20/8 core [22] to construct an internal hash function to be used in ROMix, and adopts PBKDF2

with HMAC and SHA256 to process the first and final messages. Therefore, it might be error-prone for developers to implement scrypt due to the involvement of multiple cryptographic primitives and complicated structures.

Moreover, although the internal structure ROMix is proven to be sequential memory-hard, there are no security proofs for the overall design of scrypt. Especially, the Salsa20/8 core is not collision-resistant, so it appears that scrypt can hardly be proven to be collision-resistant, which might leave scrypt certain weaknesses in some application scenarios.

5.6.2 Makwa

Makwa is a password hashing function designed by Pornin [99]. To the best of our knowledge it is the only design proposed in the PHC that adopts asymmetric-key cryptographic operations. Makwa uses an RSA/Rabin-like operation that the intermediate value x is raised to the degree of $2w + 1$, i.e., $y = x^{2w+1}$, where w is a time/work cost parameter and n is a Blum integer serving as a modulus.

Makwa is not designed to be memory-hard, and thus has very limited ability to thwart brute-force password searching based on special hardware.

5.6.3 Catena

Catena is designed by Forler *et al.*, as a provably secure password scrambler that can be used for key derivation or proof of work/space [52].

The one-wayness of Catena is guaranteed by its underlying hash function; while the security of Pleco is assured by both Keccak and the hard problem of integer factorization.

In order to avoid the random memory access pattern that makes cache-timing attacks possible [19], Catena does not employ sequential memory-hard structures like ROMix. Instead, Catena provides a new memory-hard property called λ-*memory-hard*, which focuses more on single-core settings and does not provide much resistance to parallel attacks with dedicated circuits. However, in cache-timing attacks, adversaries may need to fully or partially control victims' host machines in order to accurately measure timings, which is a difficult requirement. Thus, in our view, being sequential memory-hard is more desirable than avoiding cache-timing attacks, if these two goals are not achievable in one design of password hashing.

5.6.4 SQUASH

SQUASH is a challenge-response protocol for RFIDs designed by Shamir [114], and aims to provide provable security based on the Rabin cryptosystem. In SQUASH, a challenge is first mixed with a secret, and then processed by an optimized implementation of Rabin encryption scheme with a Mersenne number as its modulus. Our idea of combining symmetric-key and asymmetric-key cryptographic algorithms originates from the design of SQUASH.

Ouafi and Vaudenay has shown that SQUASH is insecure if the mixing function is linear [97]. PLECO and PLECTRON should not suffer from the same weakness, because the cryptographic hash function Keccak is employed as a mixing function.

5.7 Summary

In this chapter, we have proposed two provably secure password hashing algorithms, PLECO and PLECTRON. They are built upon well-studied cryptographic primitives, such as 1) a provably one-way function Rabin_n based on the hard problem of integer factorization, 2) the SHA-3 hash competition winner Keccak, and 3) the sequential memory-hard construction ROMix. We have proved that PLECO and PLECTRON inherit the security properties of Rabin_n, Keccak and ROMix, i.e., one-wayness, collision resistance and sequential memory-hardness. The designs of PLECO and PLECTRON provide two parameters *tcost* and *mcost* that can be tuned for time and memory consumptions in different application scenarios.

Chapter 6

Designing Password-less or Two-Factor Authentication Mechanisms

Two-factor authentication is an effective method for enhancing the security of entity authentication mechanisms in cloud systems, as we have introduced in Section 1.2.3. This chapter presents our security framework, Loxin, for password-less or two-factor authentication. Section 6.1 gives the detailed description of the Loxin framework, followed by the security analysis of Loxin presented in Section 6.2. We discuss several potential extensions of the Loxin framework for a wide range of applications in Section 6.3, and demonstrate that how to implement Loxin in practice to tackle the MintChip Challenge in Section 6.4. Section 6.5 gives a brief analysis of other two-factor or password-less authentication solutions. Finally, Section 6.6 summarizes the chapter.

6.1 Design of Loxin

This section describes the detailed design of Loxin, including the mechanisms to perform registration, authentication and revocation.

6.1.1 Architecture

The architecture of Loxin consists of the following components.

Loxin App
: A software application installed on users' mobile devices.

Loxin Server
: A backend server for Loxin's service, which stores the registration information about the Loxin App.

Certificate Authority (CA)
: A trusted public-key certificate authority.

Identity Provider (IDP)
: A trusted identity provider, such as an email account provider.

Push Message Service (PMS)
: A third-party service that can send notifications to users' mobile devices. Such services include Google Cloud Messaging for Android [58] and Apple Push Notification Service [11].

The adoption of the PMS makes the whole authentication process more convenient and user-friendly, but it is possible to complete the entire authentication process without the PMS. Possible extensions to achieve this are discussed in Section 6.3.

6.1.2 Registration Process

Once the Loxin App is installed, it will perform a one-time registration process as illustrated in Figure 6.1. The detailed steps are described below.

Step 1. Obtain a public-key certificate from the CA.

Step 1.1 The Loxin App generates a public-private key pair, where PK is the public key and SK is the private key. The Loxin App prompts the user to choose or enter an ID (e.g., an email address) and then sends ID and PK to the CA.

Step 1.2 The CA first communicates with the IDP and verifies the user's ID, such as sending a verification email to the claimed address. This step is simplified in Figure 6.1, since the details may vary for different providers.

Step 1.3 Once the user's ID is verified, the CA sends its signed certificate $Cert(ID, PK)$, containing both ID and PK, back to the Loxin App.

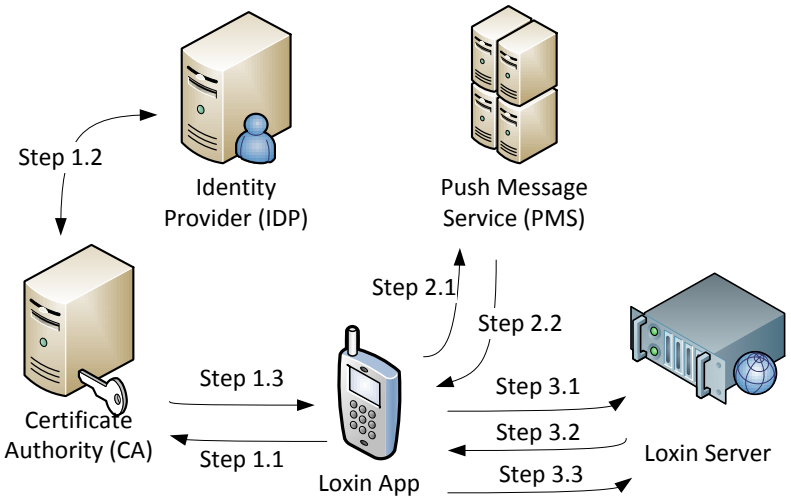

Figure 6.1: Registration process of LOXIN.

Step 1 is only required to be completed once. After that, the user can authenticate himself/herself to other cloud services by using this *ID*. Please note that the private key *SK* should be securely stored, and never be released outside the LOXIN App.

Step 2. Register to the PMS.

Step 2.1 The LOXIN App sends a registration request to the PMS.

Step 2.2 The PMS verifies the request and sends back certain credentials. These credentials can be used by other software and services to send messages to the LOXIN App via the PMS. Here we simply use a token *Tok* to represent all the PMS credentials.

Step 3. Register to the LOXIN Server securely.

Step 3.1 The LOXIN App sends a registration request, which contains $Cert(PK, ID)$ and *Tok*, to the LOXIN Server.

Step 3.2 The LOXIN Server responses with a random number R_{reg} and an expiration time T_{reg} for this request.

Step 3.3 The LOXIN App signs *ID*, *Tok*, R_{reg} and T_{reg} with its private key *SK*. The signature

$$Sig_{\text{reg}}(ID, Tok, R_{\text{reg}}, T_{\text{reg}})$$

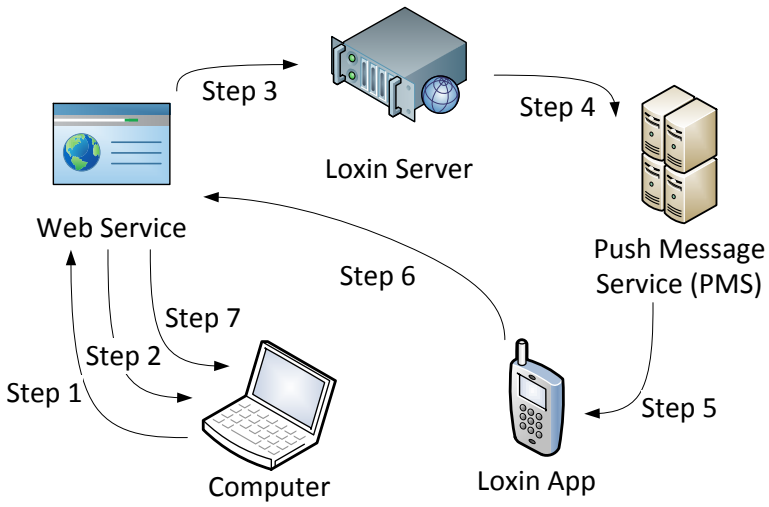

Figure 6.2: Authentication process of LOXIN.

is sent to and verified by the LOXIN Server. If the signature is valid, the LOXIN Server stores the pair (*ID*, *Tok*) into its database for later use.

Steps 2 and 3 may need to be executed multiple times for updating *Tok* when the network environment changes. However, those steps can be performed in background without users' interactions.

6.1.3 Authentication Process

Using LOXIN, users can authenticate their pre-owned identities to various cloud services even without pairing with or registering to those services first. This feature is able to shorten or remove registration processes and make cloud services more user-friendly.

When a user wants to access (e.g., log in to) a cloud service from his/her computer by using LOXIN (see Figure 6.2), a backend server of the cloud service will generate a random challenge for the user, and the LOXIN Server will forward the challenge to the LOXIN App via the PMS. Upon receiving the user's manual permission, the LOXIN App will sign the challenge with the private key *SK* and send the signature to the cloud service for verification. The authentication process is illustrated in Figure 6.2 and detailed below.

Step 1. The user enters and submits only *ID* to the cloud service.

Step 2. The cloud service generates a random number R_{auth}, an expiration time T_{auth}, and a callback address URL for this authentication request. In addition, a cryptographic hash value

$$tag = hash(ID, R_{\text{auth}}, T_{\text{auth}}, URL)$$

is computed and displayed on the user's computer. The hash value may be represented by certain formats, such as figures or colorful barcodes, other than plain strings, so it can easily be visually checked by the user.

Step 3. The cloud service sends ID, R_{auth}, T_{auth}, and URL to the LOXIN Server.

Step 4. The LOXIN Server searches ID in its database in order to retrieve the corresponding Tok. Then the LOXIN Server uses Tok to send R_{auth}, T_{auth}, and URL to the PMS.

Step 5. The PMS forwards R_{auth}, T_{auth}, and URL to the user's LOXIN App.

Step 6. The LOXIN App recomputes the hash value tag based on the received ID, R_{auth}, T_{auth}, and URL. The LOXIN App prompts the user to verify the correctness of basic authentication information, and compare the figures or barcodes shown on the computer and the LOXIN App (see Figure 6.3 for an example).

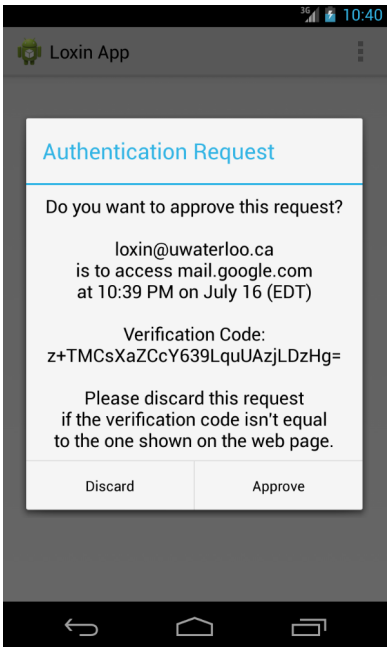

Figure 6.3: An example confirmation dialog of the LOXIN App.

Once *tag* and other information are approved, the LOXIN App computes the signature

$$Sig_{\text{auth}}(ID, R_{\text{auth}}, T_{\text{auth}}, URL)$$

with the private key SK, and then sends Sig_{auth} and $Cert(ID, PK)$ to the cloud service's address URL.

Step 7. After verifying $Cert(ID, PK)$ and Sig_{auth} together with R_{auth} and T_{auth}, the cloud service grants access to the user.

Requiring users to confirm the hash tags is to "bind" together different communication channels, i.e., the web requests made from the computer and the push message received by the LOXIN App; otherwise the user may not have an easy way to differentiate legitimate authentication requests and the requests made by adversaries. Section 6.5.5 discusses an example that fails to bind different communication channels and makes attacks possible.

As the step to manually confirm the hash tags may not be very user-friendly, we can choose certain alternative approaches to bind communication channels. One potential solution is splitting Step 6 of the authentication process into two steps. Firstly, once an authentication request is received by the LOXIN App and approved by the user, the LOXIN App sends an initial message *without* the signature Sig_{auth} to the address URL. When the cloud service provider receives the initial message, the cloud service interface, e.g., a web page, shown on the user's computer should be updated in a specific way in order to indicate that the cloud service has received the initial message. The second step is that after confirming the cloud service interface update, the user instructs the LOXIN App to send the signature Sig_{auth} with other related information to the cloud service provider. If the user does not see the interface update on his/her computer, he/she should not confirm the authentication request or continue to send the signature. In this way, different communication channels are bound by the two steps, which may make the authentication process more user-friendly. However, this splitting-step method might pose other security concerns, e.g., URL must be unique and infeasible to be guessed, which requires more considerations, so this chapter still focuses on the setting that users will always faithfully compare the displayed hash tags and verify authentication requests.

6.1.4 Revocation Mechanisms

If a user's phone is lost, the private key SK stored in the LOXIN App might be compromised. In this case, the user needs to contact the CA to revoke the certificate of the corresponding

public key *PK*. For example, if the CA allows only one certificate for each *ID*, the user may go through the registration process (see Section 6.1.2) again to revoke the old certificate.

Contacting the CA to revoke the lost certificate may be time-consuming, and the user's email account may be compromised as well if the mobile device is lost. One potential solution to expedite the revocation process is authenticating to the LOXIN Server by using certain biological information, such as the fingerprint authentication information provided by iPhone's Touch ID [12]. Once the LOXIN Server receives the revocation request, it will block further authentication requests associated with the *PK*. The biological information may also be used by the CA to verify users in order to revoke certificates.

The other possible method is generating a second pair of public and private keys, *PK'* and *SK'*, during the registration process. This second key pair should be stored out of the mobile device, e.g., printing on a paper, for security considerations. If the user's primary secret key *SK* may be leaked, the user can authenticate his/her identity by using *PK'* and *SK'* to the LOXIN server or the CA.

In order to minimize the risk that the user's private key is used by adversaries, certain countermeasures should be deployed, e.g., requiring a fingerprint scan on iPhone's Touch ID sensor or a short PIN to access the LOXIN App, and limiting the number of retrials. Please note that adding a PIN will make the application less convenient, but it is still much more user-friendly than remembering and entering long passwords.

6.2 Security Analysis

This section aims to analyze the security of LOXIN. In addition, several methods are provided to further enhance the security of LOXIN.

6.2.1 Defeating Man-in-the-Middle Attacks

In order to guarantee that the *tag* displayed on the computer is really the one generated by the cloud service provider, the Internet connection between the cloud service and the user's computer should be well protected by certain secure transport layer such as TLS. Next, *tag* shown by the LOXIN App will be compared by the user with the one shown on the computer, and thus ensure both R_{auth} and T_{auth} are not replaced by any adversary in the middle. As long as the *tag* shown on the web page is genuine and matches with the one displayed on the LOXIN App, disclosure of the request information transmitted in

the authentication process will not affect the authenticity of the authentication process. Therefore, man-in-the-middle attacks cannot gain benefits for attackers.

6.2.2 Defeating Replay Attacks

Both registration and authentication processes involve a random number to prevent replay attacks. The random numbers are recommended to be at least 128 bits, such that it is infeasible for attackers to obtain two requests with a same random number in order to re-send the eavesdropped public-key signatures to impersonate the user.

Besides the random number, an expiration time is also included in the registration and authentication processes, which will keep the whole system safe even if a random number collision occurs in the long term.

6.2.3 Defeating Server Compromises

Since the private key SK never leaves the LOXIN App, any backend server or cloud service does not have the knowledge of SK. Therefore, as long as the IDP and CA are secure, even if LOXIN's backend servers are compromised, attackers will not be able to authenticate themselves to other cloud services.

6.2.4 Further Security Enhancements

One method to enhancing the security of LOXIN is to sign the user's ID and public-key PK by multiple CAs. In this case, adversaries have to compromise all these CAs to generate a fake certificate. Additionally, if one CA does not update its revocation list promptly, cloud service providers can still check with other CAs. The other benefit is that the entire LOXIN service will not be controlled by a single CA, also known as vendor lock-in, since CAs work equivalently in the LOXIN framework.

The other security enhancement is the public-key pinning, i.e., users' certificates are required to be signed by a small group of specific CAs. This will prevent dishonest CAs, whose root certificates have already been embedded in various operating systems, from creating fake certificates for LOXIN.

If any users or organizations need a higher level of security, e.g., for protecting business secrets, hardware security modules (HSMs) can be used with LOXIN. A HSM exposes only

necessary interfaces, such as signature computation and verification, to operating systems and applications, which will minimize the possibility of leaking the private key *SK*.

6.2.5 Security Limitation

As we mentioned before, the LOXIN system gives the ability of using one user's pre-owned *ID* to access other cloud services, and the *ID* has to be authenticated by the IDP during the registration process (see Figure 6.1). For example, if one uses an email address as *ID*, the address may be authenticated via the email service provider to the CA. Therefore, the security of the LOXIN system still relies on the trustworthiness of the IDP. In this sense, the security of LOXIN is similar to that of OpenID. Nevertheless, LOXIN allows cloud service providers to directly verify the users' signatures by public-key certificates without the help from the IDP, which improves scalability and reduces network protocol latency. Moreover, the protocols of LOXIN enable a user to securely complete authentication from multiple devices easily with one smart phone.

6.3 Application Extensions

This section presents several methods to extend the original design of LOXIN for a wide range of applications.

6.3.1 Two-Factor Authenticator

LOXIN is fully compatible with traditional password-based authentication schemes, which means that even if users initially do not trust the security of the LOXIN system, they can still use LOXIN as a convenient security enhancement, i.e., a two-factor authenticator.

This may help smoothing the adoption process of LOXIN in early stages. Service providers can first add LOXIN as a two-factor security enhancement, and then give users the option to use LOXIN as the sole authentication method.

6.3.2 Local Authentication

Typing passwords is particularly painful on the relatively small screen of a smart phone. The LOXIN App can also be used to authenticate other applications installed on the smart

phone. In this special case, the authentication process of LOXIN can be executed locally without involving the LOXIN Server or PMS. An application can broadcast a local authentication request within the phone, and once the LOXIN App receives the request it can reply a proper signature upon the user's approval.

6.3.3 Authentication via Barcode

If the LOXIN Server or PMS is offline, the authentication request from the cloud service will not reach the user in time. In this case, the cloud service can display a barcode (e.g., a QR code) to the user on the computer, which contains all the necessary information about the request. After scanning the barcode, the LOXIN App can send the authentication signature to the cloud service directly. This method prevents the whole authentication process from the potential single point of failure of the LOXIN Server.

6.3.4 Pairing without ID

It is possible to use the LOXIN service even without first telling the user's *ID* to cloud service providers. For example, after scanning the barcode as described in the previous subsection, the LOXIN App will send the user's public-key certificate along with the signature, and then the cloud service can retrieve *ID* from the certificate. Thus the user does not need to manually enter *ID* during the entire authentication process. In the original design described in Section 6.1.3, it is possible to utilize some other factors, such as geographic and network information, to pair the LOXIN App with the cloud service.

6.4 LOXIN in Practice – Tackling the MintChip Challenge

In this section, we apply the LOXIN security framework to build a password-less mobile payment solution called EASYCHIP for tackling the real-world MintChip Challenge [89] organized by the Royal Canadian Mint. With LOXIN in place, a user can complete online transactions without creating additional accounts with multiple merchants, thereby offering an innovative password-less online payment service.

6.4.1 The MintChip Challenge

In 2012, the Canadian federal government announced in its budget that it would withdraw the penny from circulation in the fall of 2012. As a quick response, the Royal Canadian Mint unveiled its digital alternative called MintChip [89] to coinage and small bank denominations, and simultaneously launched the MintChip Challenge contest to encourage the development of novel applications based on MintChip.

A MintChip, as illustrated in Figure 6.4, is a secure smart card chip that can be encapsulated into different form factors (e.g., a MicroSD card) for easier connections to computers and mobile devices. The MintChip securely holds electronic money and enables a protocol to transfer it from one chip to another. The main goal of the MintChip is to facilitate small-value transactions, such as micro-transactions (under $10) and nano-transactions (under $1). Unlike existing digital wallets, such as Google Wallet [54] and Apple Pay [10], where customers' financial information (e.g., credit/debit card) is stored into an embedded secure element or in the cloud, MintChip does not have any link to your bank account or credit card and no personal data is exchanged during a transaction.

Figure 6.4: A pair of MintChip's (centre) and accessories from the Royal Canadian Mint.

6.4.2 The EasyChip Solution

To tackle the MintChip Challenge, we have developed EasyChip [122], an Android application for password-less mobile payment based on the Loxin security framework presented in Section 6.1. Using the EasyChip application on a smart phone, a password-less payment process works as described below.

Registration

In the Loxin framework, the Loxin App needs to first obtain a public-key certificate from the CA. However, the MintChip inside a smart phone has already contained a unique 64-bit MintChip ID, a preloaded private/public RSA key pair, and the associated X.509 public-key certificate issued by the MintChip CA. Therefore, Steps 1.1 – 1.3 in the Loxin registration procedure can be omitted. Secondly, the EasyChip App selects/creates an existing/new email account and registers it to Google Cloud Messaging for Android for the push message service. Finally, the EasyChip App registers to the backend server with the email account, the MintChip ID, the MintChip certificate, and the push message service token as described in Steps 3.1 – 3.3 of the Loxin security framework.

Authentication and Payment

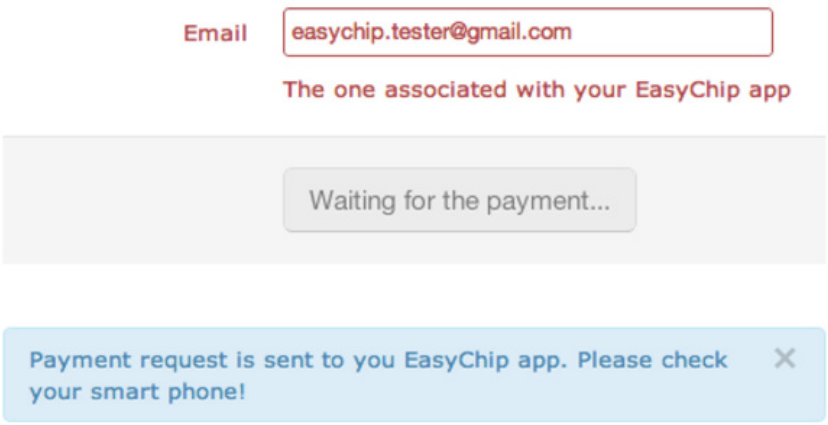

Figure 6.5: A customer has submitted his/her email address.

A complete MintChip payment always involves two MintChip devices, namely a sender and a receiver. Moreover, the receiver's MintChip ID must be known by the sender. When a customer (i.e., a sender) wants to purchase a product from a merchant website (i.e., a receiver), the customer first enters the email address associated with the EasyChip App, as shown in Figure 6.5.

The merchant's web server, which is equipped with another MintChip, generates a MintChip Request message that contains the information such as the receiver's MintChip ID, the amount to pay, a URL specifying where the payment should be sent to, a random

challenge, etc. The MintChip Request message and the customer's email address will be sent to the LOXIN Server. Upon receiving the message, the LOXIN Server looks up its database with the customer's email address and retrieves the push message service token. Then the LOXIN Server pushes the MintChip Request message to the customer's smart phone through the PMS, as shown in Figure 6.6.

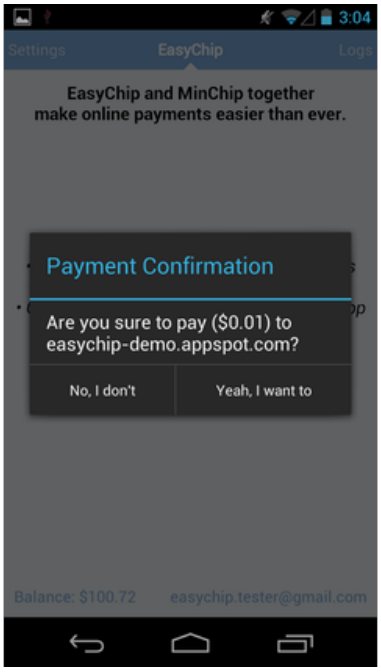

Figure 6.6: The payment requires the approval by the user.

When the customer confirms the payment request, the MintChip inside the customer's smart phone will immediately generate a signed MintChip Value message using the RSA signature scheme and send it back to the merchant's web server. After verifying the received MintChip certificate and digital signature, the payment will be processed (see Figure 6.7). Note that the entire authentication and payment processes follow the LOXIN security framework, and the customer does not need to enter any password.

6.5 Comparisons with Other Authentication Mechanisms

In this section, we analyze several other authentication mechanisms, and also provide their comparisons with LOXIN.

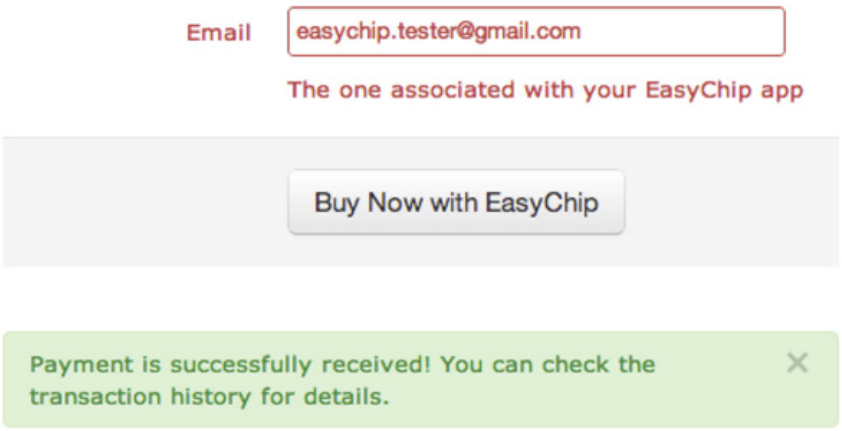

Figure 6.7: The online transaction has completed.

6.5.1 RSA SecurID

RSA SecurID is a well-established product in the two-factor authentication market, which is a hardware token with a small screen showing a pseudorandom authentication code in every minute [106]. Each RSA SecurID shares a secret seed with its backend server. When a user submits the authentication code to a cloud service, the service provider will compute the number based on their own knowledge of the secret seed and then compare it with the one submitted by the user.

If the servers of RSA SecurID are compromised, attackers can compute any pseudorandom authentication codes after obtaining their secret seeds. In fact, this kind of incidents did happen in 2011 [37], which renders RSA SecurID less effective to serve as a secure two-factor authentication mechanism. Moreover, RSA SecurID also has a usability issue and users have to carry the extra hardware device. In addition, different cloud services usually do not share an identical secret seed, so user may be required to have multiple devices associated with various service providers.

6.5.2 Google Authenticator

Google Authenticator [64] is a software solution to the usability issue of RSA SecurID. It replaces the hardware device of RSA SecurID by a software application on users' mobile devices, and can be paired with many service providers such that users do not need to carry multiple devices.

However, Google Authenticator still shares seeds with its backend servers, and is required to be manually paired with each service provider similarly as RSA SecurID, which is not user-friendly when compared with LOXIN.

6.5.3 Kerberos

Kerberos is a symmetric-key cryptography based protocol that allows users authenticate their identities to services by the help of a central Kerberos server [95]. A *ticket* will be issued by the central server for a specific service when the user wants to access the service.

Kerberos apparently suffers from single point of failure of the central Kerberos server. In addition, although a public-key cryptography based initial authentication extension is proposed in [129], the *ticket* issued in Kerberos system is still produced by symmetric-key algorithms. Thus once the database of the Kerberos server is compromised, the credentials of all users will be in danger.

6.5.4 Pico

Pico is a hardware solution proposed by Stajano in 2011 [117], which serves as a replacement of password-based authentications. Pico is recommended to be a dedicated device with capabilities such as a camera and a radio. It is hard to manufacture, and is required to be carried by users all the time. Moreover, Pico has to be paired with each application in a similar way as RSA SecurID and Google Authenticator.

6.5.5 Twitter's Two-Factor Authentication

Recently, Twitter upgraded its mobile applications to support a public-key cryptography based two-factor authentication solution [116], which has a similar idea as LOXIN in the sense that the web server sends a login challenge to the user and requires it to be signed by the private key stored in the smart phone application.

As mentioned in [14], the design of Twitter's two-factor authentication mechanism has a security hazard that users cannot tell the differences between the fake login requests initiated by adversaries and the real ones by the users themselves, since the smart phone application does not provide the user with detailed information about login requests. The hash value *tag* used in the LOXIN system can be adopted to defeat this kind of attacks. Moreover, the public key is only paired with Twitter, which is similar to the method of

Pico. To provide single-sign-on service to other service providers, the public key needs to be properly signed by trusted third-party CAs.

6.5.6 Mozilla Persona

Persona (formerly BrowserID) is a decentralized single-sign-on system developed by Mozilla for users and websites to release the burden of creating and managing passwords [94]. Persona adopts users' email addresses as identities and issues public-key certificates for these emails.

However, the design of Persona aims to provide in-browser solution and stores the public-key certificate in the local space of a browser. Therefore, to use on multiple devices, Persona may need to be set up many times, which is not as convenient as LOXIN. With the help of push message services, LOXIN allows a user to store his/her private key in a smart phone and access many services on multiple computers or devices.

6.5.7 PhoneAuth

PhoneAuth is a user-friendly two-factor authentication mechanism proposed in 2012 [38]. The authentication request is automatically signed by the smart phone application, if the user's smart phone is present and can be connected to the computer via Bluetooth. The whole two-factor authentication process does not need the user's interaction.

However, PhoneAuth requires the web browser to be capable of sending data to the user's smart phone via a Bluetooth connection. The authors of [38] managed to achieve this function by developing an extension for the Chromium web browser. Regular web browsers without any modifications do not have such abilities, and it would be dangerous to open a web interface of physically accessing users' smart phones.

6.5.8 Duo Push

Duo Push is a commercial software application developed by Duo Security [46], which aims to provide a two-factor authentication with push message capabilities. However, the design details of Duo Push are not disclosed. Moreover, the authentication status of a user in Duo Push depends on the response from the verification servers of Duo Security, which makes Duo Push unsuitable for replacing password-based authentication solutions used by other companies and organizations. Furthermore, the systems integrated with Duo Push may

have the single point of failure, as users will not be able to access cloud services if the verification servers of Duo Security are not working properly or being compromised.

6.6 Summary

In this chapter, we have proposed an authentication framework called Loxin. We have demonstrated that Loxin is secure against man-in-the-middle attacks and replay attacks. In particular, even if the servers of Loxin are compromised by attackers, the private keys of users are still safe and thus attackers cannot impersonate the users. This special feature makes Loxin an attractive security solution for password-less or two-factor authentication. Several methods have also been proposed to extend Loxin for using in different application scenarios, and for avoiding single point of failure as well as vendor lock-in. We have also developed EasyChip, an Android application following the Loxin security framework, to demonstrate the power of Loxin for building a real-world password-less mobile payment solution.

Chapter 7

Concluding Remarks and Future Work

This chapter concludes the thesis and discusses several potential topics for future research.

7.1 Conclusions and Our Contributions

The salient features of cloud systems, such as scalability, flexibility, reliability, and low costs, have attracted a huge number of customers. More and more organizations and companies are outsourcing their computation, storage and network needs to cloud-based solutions. Due to the nature of cloud computing, such as resource sharing/pooling and web-based remote connections, security plays an important role in cloud system designs. Cloud service providers need to protect authenticity and confidentiality of customers' data transmitted to and stored in the cloud, and prevent unauthorized access of customers' resources. This thesis has investigated various authentication and encryption algorithms that protect cloud systems, including

- modes of operation for data encryption and authentication,
- block ciphers for encryption,
- password hashing algorithms for password-based authentication and key derivation functions, and
- password-less or two-factor authentication mechanisms.

Table 7.1: Main functionalities of the studied algorithms.

Algorithms	Authentication	Encryption
Modes of operation	✓	✓
Block ciphers		✓
Password hashing	✓	
Password-less/two-factor	✓	

Table 7.2: Positions of our work in the thesis.

Algorithms	Analysis	Design
Modes of operation	Chapter 3	Section 2.3
Block ciphers	Chapter 4	
Password hashing		Section 5.2
Password-less/two-factor	Section 6.5	Section 6.1

The main functionalities of these algorithms are summarized in Table 7.1, and the positions of our work on either analysis or design of these algorithms are listed in Table 7.2.

We list our contributions for each topic as follows.

- Authenticated encryption modes of operation:
  - We have proposed a simple change to repair GCM in order to avoid the security proof flaw discovered by Iwata *et al.* The security bounds of the revised algorithm, called LGCM, are tighter than the original ones of GCM by a factor of about 2^{20} for non-96-bit nonces.
  - We have presented two alternative methods for implementing LGCM such that LGCM has better resistance against timing-based side-channel attacks.
  - We have demonstrated that hash collisions are not necessary for constructing forgeries of certain polynomial-based MAC schemes like the one adopted in GCM. This discovery removes certain restrictions in the attacks proposed by Procter and Cid.
  - We have proven that all authentication key sets with no less than two elements are weak key classes for GCM-like polynomial-based MAC schemes, which is an extension to Procter and Cid's analysis result.
  - We have shown that due to a special structure of GCM, these forgery attacks can be turned into birthday attacks, which will significantly increase their success probabilities.

  - We have indicated that these forgery attacks can still work even if GCM is changed to the MAC-then-Enc paradigm, as one of the methods proposed by the previous studies, for further securing GMAC.

  Considering the birthday attacks together with the flaw found by Iwata *et al.*, we recommend that we should avoid using GCM with non-96-bit nonces. However, if non-96-bit nonces are preferred in certain applications, our revised algorithm LGCM is recommended.

- Block ciphers:

  - We have proposed a new framework, namely multidimensional meet-in-the-middle attack, to analyze symmetric-key ciphers by segmenting algorithms into consecutive sub-ciphers. This framework is suitable for lightweight ciphers with simple key schedules and block sizes smaller than key lengths.
  - Following this framework, we have devised new attacks that can recover the master keys of 175-round KATAN32, 130-round KATAN48, and 112-round KATAN64 faster than exhaustive search. These new attacks can reach more rounds than the existing attacks.
  - We have also provided new attacks on 115-round KATAN32 and 100-round KATAN48 in order to demonstrate that this new kind of attacks can be more time-efficient and memory-efficient than the existing ones.

- Password hashing algorithms:

  - We have proposed two practical password hashing algorithms, Pleco and Plectron. They are built upon well-studied cryptographic algorithms and combine the advantages of symmetric-key and asymmetric-key primitives.
  - We have given the security proofs for the one-wayness and collision resistance of Pleco and Plectron.
  - We have designed both password hashing algorithms to be sequential memory-hard, in order to thwart large-scale parallel password searching using dedicated hardware, such as GPUs, FPGAs, and ASICs.
  - We have also proposed several variants of Pleco and Plectron with better security bounds or more efficient software implementations.

- Password-less or two-factor authentication:

– We have proposed Loxin, a solution for the entity authentication of cloud systems and web applications, which enables users to access multiple services by using pre-owned identities without creating or submitting any passwords.
– We have shown that Loxin is invulnerable to common attacks such as man-in-the-middle and replay attacks. In particular, servers in the Loxin system do not generate or possess the authentication credentials of users, so compromised Loxin servers will not leak users' credentials.
– Loxin is compatible with existing password-based authentication systems, and thus it can serve as a two-factor authentication mechanism.
– We have given an analysis of other popular two-factor/password-less authentication mechanisms used for cloud systems, as well as comparisons with our design Loxin.

7.2 Future Research

This section discusses several ideas, which we have gotten during the process of finishing the work in this thesis, for potential topics of our future research.

7.2.1 Authenticated Encryption Modes of Operation

Due to the adoption of GCM in a number of widely used protocols, e.g., TLS v1.2, IEEE 802.1AE and IPsec, it is important for researchers to find practical solutions to avoid the issues of GCM in real-world applications. A competition is currently ongoing for collecting the ideas and designs of new authenticated encryption algorithms [1], but it would still take a long time before new designs are intensively analyzed and then ready for practical applications. Therefore, repairing security issues of existing algorithms is still important for the current cloud system designs and implementations that may stay for a long time. We are curious and eager to discover simpler or more efficient methods than LGCM for repairing GCM.

Besides block ciphers, stream ciphers are also important encryption algorithms widely used in practical applications. Stream ciphers have been adopted in many real-time communication systems, e.g., 4G LTE [90], due to the simple and efficient hardware implementations. The stream cipher ChaCha [21] has been implemented in the web servers and software of Google as an efficient replacement for the block cipher AES [32]. However, to

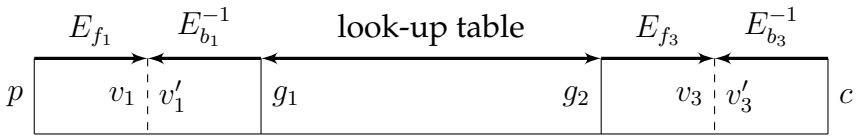

Figure 7.1: Multidimensional MITM attacks with look-up tables.

the best of our knowledge, there is not a widely-accepted authenticated encryption mode of operation for stream ciphers. Stream ciphers are designed for protecting confidentiality but not authenticity, so in practice stream ciphers are usually used with general-purpose authentication algorithms such as HMAC, or block cipher modes of operation such as Poly1305. An authenticated encryption mode specifically designed for stream ciphers may have better efficiencies and security bounds.

7.2.2 Block Ciphers

For MD-MITM attacks with dimensions larger than two, there are portions of the attacks that can be pre-computed. For example, the two ends of the middle portion for a 3D-MITM attack are both guessed values. Thus we can build a look-up table for the intermediate computations off-line without any knowledge about the plaintexts and ciphertexts, which is illustrated in Figure 7.1. Especially, we can use any approach, not only limited to MITM methods, to construct the look-up table in Figure 7.1.

We can also analyze other lightweight block ciphers as future work. We can consider KATAN's sibling block cipher family KTANTAN32/48/64 [33], which has the same round functions as KATAN but a different key scheduling algorithm. KTANTAN has been analyzed by many researchers [120, 29]. By refining our matching techniques, we may be able to construct more efficient attacks on KTANTAN, similar as the ones on KATAN.

7.2.3 Password Hashing Algorithms

In order to fully utilize the memory-hardness property of ROMix, the internal hash function $\mathcal{H}_n$ should be as fast as possible, since during a fixed time period the total amount of memory that can be consumed is limited by the computational speed of $\mathcal{H}_n$. The current design of $\mathcal{H}_n$ is based on modular squaring of big integers, so it may not be fast enough for certain devices with constrained CPU power. We have proposed several modified designs

of $\mathcal{H}_n$ that have better software efficiency in Section 5.4.4. However, the security of these potential solutions needs further investigations.

Another potential method to improve the computational speed of our password hashing designs is refining the steps of ROMix. The current design of ROMix can be viewed as a combination of two phases. The first phase fills the memory with the hash function outputs, and the second phase randomly accesses the values in the memory. It is possible to merge these two phases, by randomly accessing the previous outputs when filling new memory locations. This method may save almost half of the overall computation time, but it needs careful investigations and proofs of its memory-hard property.

It is encouraging to design password hashing algorithms by combining asymmetric-key and symmetric-key primitives, since the combined algorithms can offer provable security and potential server-specific computational shortcuts. We expect more password hashing designs consisting of both asymmetric-key and symmetric-key components to appear.

7.2.4 Password-less Entity Authentication

As we have discussed in Section 6.1.3, there are alternative methods, such as a splitting-step method for request approval, that are more user-friendly than manually comparing hash tags, in order to bind separated communication channels and avoid potential attacks. We would like to carefully analyze these new methods and extend the framework of Loxin to include them for improving Loxin's overall usability without compromising security.

APPENDICES

Appendix A

Partial-Matching Details for the Attacks on KATAN48 and KATAN64

The detailed steps of the partial matching in the 2D-MITM attack on KATAN48, described in Section 4.4.1, are given as follows. The notations follow the ones used in Section 4.3.1.

Rd.	a	b	L1	L2
second backward phase				
92	0	0	0000000000000000000	00000000000000000000000000000
91	1	1	0000000000000000011	00000000000000000000000000011
90	0	0	0000000000000001101	00000000000000000000000001100
89	0	0	0000000000000110110	00000000000000000000000110000
88	0	0	0000000000011011011	00000000000000000000011000001
87	0	0	0000000001101101101	00000000000000000001100000111
86	0	0	0000000110110110111	00000000000000000110000011110
85	0	0	0000011011011011111	00000000000000011000001111001
84	0	0	0001101101101111111	00000000000001100000111100111
83	0	0	0110110110111111111	00000000000110000011110011111
82	1	0	1011011011111111111	00000000011000001111001111111
81	0	1	1101101111111111111	00000001100000111100111111111
80	0	0	0110111111111111111	00000110000011110011111111111
79	0	0	1011111111111111111	00011000001111001111111111111
78	0	0	1111111111111111111	01100000111100111111111111111
77	0	0	1111111111111111111	10000011110011111111111111111
76	1	1	1111111111111111111	00001111001111111111111111111

```
75   0 0  1111111111111111111  00111100111111111111111111111
74   0 0  1111111111111111111  11110011111111111111111111111
73   0 0  1111111111111111111  11001111111111111111111111111
72   1 1  1111111111111111111  00111111111111111111111111111
second forward phase
71   0 0  0000000000000000000  00000000000000000000000000000
matching
2 bits    1111111111111111111  00111111111111111111111111111
```

The partial-matching steps for the 2D-MITM attack on KATAN64 stated in Section 4.4.2 are as follows.

```
Rd.  a b  L1                          L2
first forward phase
39   0 0  000000000000000000000000  00000000000000000000000000000000000000
40   1 1  111000000000000000000000  11100000000000000000000000000000000000
41   0 0  000111000000000000000000  00011100000000000000000000000000000000
42   0 0  000000111000000000000000  00000011100000000000000000000000000000
43   0 0  110000000111000000000000  00000000011100000000000000000000000000
44   0 0  001110000000111000000000  11100000000011100000000000000000000000
45   0 0  111001110000000111000000  11011100000000011100000000000000000000
46   0 1  111111001110000000111000  10111011100000000011100000000000000000
first backward phase
58   1 1  000000000000000000000000  00000000000000000000000000000000000000
57   1 1  000000000000000000000111  00000000000000000000000000000000000111
56   1 1  000000000000000000111111  00000000000000000000000000000000111111
55   1 1  000000000000000111111111  00000000000000000000000000000111111111
54   1 1  000000000000111111111111  00000000000000000000000000111111111111
53   1 1  000000000011111111111111  00000000000000000000000011111111111111
52   1 1  000000011111111111111111  00000000000000000000011111111111111111
51   1 1  000111111111111111111111  00000000000000000011111111111111111111
50   1 1  011111111111111111111111  00000000000000011111111111111111111111
49   1 1  111111111111111111111111  00000000000011111111111111111111111111
48   1 1  111111111111111111111111  00000000111111111111111111111111111111
47   1 1  111111111111111111111111  00000111111111111111111111111111111111
matching
2 bits    111111111111111111111111  10111011111111111111111111111111111111
```

Appendix B

Test Vectors for PLECO and PLECTRON

For testing and reference, we give the following input parameters and their corresponding hashing output of PLECTRON, using the Mersenne number $2^{2137}-1$ as the modulus. Please note that the hexadecimal numbers for the entries *salt*, *pass* and *tag* represent byte strings, e.g., `546865` means an ASCII string *The*. Long strings are written in multiple lines.

salt	4c880aa553669c3869f62b389c2c3499
pass	54686520717569636b2062726f776e20
	666f78206a756d7073206f7665722074
	6865206c617a7920646f67
tcost	2
mcost	1024
hsize	256
tag	7969ad4aae09ba48e61cc5e348f1de39
	c15475d69eee42cffe8770a88f2f3e93

Appendix C

Names of the Proposed Password Hashing Algorithms

Pleco or *Plecostomus* is the name for a kind of catfish that is very popular among aquarists, as Pleco fish help keeping water clean. The word Plecostomus means folded mouth. *Plectron* refers to a small piece of metal or plastic that is used to plunk musical instruments.

References

[1] CAESAR – Competition for Authenticated Encryption: Security, Applicability, and Robustness. `http://competitions.cr.yp.to/caesar.html`. Accessed November 2014.

[2] Dogecoin. `http://dogecoin.com/`. Accessed November 2014.

[3] Litecoin – Open source P2P digital currency. `https://litecoin.org/`. Accessed November 2014.

[4] Password Hashing Competition. `https://password-hashing.net/index.html`. Accessed November 2014.

[5] Martin R. Albrecht and Gregor Leander. An all-in-one approach to differential cryptanalysis for small block ciphers. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2012.

[6] Amazon Web Services, Inc. Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting. `http://aws.amazon.com/ec2/`. Accessed November 2014.

[7] Amazon Web Services, Inc. AWS Elastic Beanstalk – Amazon Web Services. `http://aws.amazon.com/elasticbeanstalk/`. Accessed November 2014.

[8] Kazumaro Aoki and Yu Sasaki. Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2009.

[9] Kazumaro Aoki and Kan Yasuda. The security and performance of "GCM" when short multiplications are used instead. In Miroslaw Kutylowski and Moti Yung, editors, *Information Security and Cryptology - 8th International Conference, Inscrypt 2012, Beijing, China, November 28-30, 2012, Revised Selected Papers*, volume 7763 of *Lecture Notes in Computer Science*, pages 225–245. Springer, 2012.

[10] Apple Inc. Apple Pay. `https://www.apple.com/apple-pay/`. Accessed November 2014.

[11] Apple Inc. Apple Push Notification Service. `https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html`. Accessed November 2014.

[12] Apple Inc. Use Touch ID on iPhone and iPad. `http://support.apple.com/en-us/HT5883`. Accessed November 2014.

[13] Jean-Philippe Aumasson and Daniel J. Bernstein. SipHash: A fast short-input PRF. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *Lecture Notes in Computer Science*, pages 489–508. Springer, 2012.

[14] Authy, Inc. Thoughts on Twitter's new two-factor authentication. `http://blog.authy.com/twitter`, 2013. Accessed November 2014.

[15] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management, part 1: general (revision 3). NIST Special Publication 800-57, 2012.

[16] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer, 1994.

[17] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *J. Comput. Syst. Sci.*, 61(3):362–399, 2000.

[18] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual*

International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993.

[19] Daniel J. Bernstein. Cache-timing attacks on AES. `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`, 2005. Accessed November 2014.

[20] Daniel J. Bernstein. The Poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005.

[21] Daniel J. Bernstein. ChaCha, a variant of Salsa20. In *Workshop Record of SASC*, volume 8, 2008.

[22] Daniel J. Bernstein. The Salsa20 family of stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, pages 84–97. Springer, 2008.

[23] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak SHA-3 submission. Submission to NIST (Round 3), 2011.

[24] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak implementation overview, version 3.2. `http://keccak.noekeon.org/Keccak-implementation-3.2.pdf`, 2012. Accessed November 2014.

[25] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[26] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2000.

[27] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December*

4-8, 2011. Proceedings, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.

[28] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[29] Andrey Bogdanov and Christian Rechberger. A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2010.

[30] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 553–567. IEEE Computer Society, 2012.

[31] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[32] Elie Bursztein. Speeding up and strengthening HTTPS connections for Chrome on Android. `http://googleonlinesecurity.blogspot.com/2014/04/speeding-up-and-strengthening-https.html`, 2014. Accessed November 2014.

[33] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded*

Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[34] Abhijit Choudhury, David McGrew, and Joseph Salowey. AES Galois Counter Mode (GCM) cipher suites for TLS. RFC 5288, available at `https://tools.ietf.org/html/rfc5288`, 2008.

[35] Luke St. Clair, Lisa Johansen, William Enck, Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Trent Jaeger. Password exhaustion: Predicting the end of password usefulness. In Aditya Bagchi and Vijayalakshmi Atluri, editors, *Information Systems Security, Second International Conference, ICISS 2006, Kolkata, India, December 19-21, 2006, Proceedings*, volume 4332 of *Lecture Notes in Computer Science*, pages 37–55. Springer, 2006.

[36] Nicolas T. Courtois. Algebraic complexity reduction and cryptanalysis of GOST. Cryptology ePrint Archive, Report 2011/626, 2011. `http://eprint.iacr.org/`.

[37] Art Coviello. Open letter to RSA customers. `https://www.sec.gov/Archives/edgar/data/790070/000119312511070159/dex991.htm`, 2011. Accessed November 2014.

[38] Alexei Czeskis, Michael Dietz, Tadayoshi Kohno, Dan S. Wallach, and Dirk Balfanz. Strengthening user authentication through opportunistic cryptographic identity assertions. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 404–414. ACM, 2012.

[39] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[40] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) protocol version 1.2. RFC 5246, available at `https://tools.ietf.org/html/rfc5246`, 2008.

[41] Whitfield Diffie and Martin E. Hellman. Exhaustive cryptanalysis of the NBS data encryption standard. *IEEE Computer*, 10(6):74–84, 1977.

[42] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances*

in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings, volume 7417 of *Lecture Notes in Computer Science*, pages 719–740. Springer, 2012.

[43] Itai Dinur, Orr Dunkelman, and Adi Shamir. Improved attacks on full GOST. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 9–28. Springer, 2012.

[44] NIST Computer Security Division. The SHA-3 cryptographic hash algorithm competition. `http://csrc.nist.gov/groups/ST/hash/sha-3/`. Accessed November 2014.

[45] Benedikt Driessen. Eavesdropping on satellite telecommunication systems. Cryptology ePrint Archive, Report 2012/051, 2012. `http://eprint.iacr.org/`.

[46] Duo Security, Inc. Duo Push: One-tap authentication. `https://www.duosecurity.com/duo-push`. Accessed November 2014.

[47] Markus Dürmuth, Tim Güneysu, Markus Kasper, Christof Paar, Tolga Yalçin, and Ralf Zimmermann. Evaluation of standardized password-based key derivation against parallel processing platforms. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, volume 7459 of *Lecture Notes in Computer Science*, pages 716–733. Springer, 2012.

[48] Morris J. Dworkin. Recommendation for block cipher modes of operation: Methods and techniques. NIST Special Publication 800-38A, 2001.

[49] Morris J. Dworkin. Recommendation for block cipher modes of operation: the CCM mode for authentication and confidentiality. NIST Special Publication 800-38C, 2004.

[50] Daniel W. Engels, Markku-Juhani O. Saarinen, Peter Schweitzer, and Eric M. Smith. The Hummingbird-2 lightweight authenticated encryption algorithm. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*, volume 7055 of *Lecture Notes in Computer Science*, pages 19–31. Springer, 2011.

[51] Niels Ferguson. Authentication weaknesses in GCM. Comments submitted to NIST Modes of Operation Process, `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/CWC-GCM/Ferguson2.pdf`, 2005. Accessed November 2014.

[52] Christian Forler, Stefan Lucks, and Jakob Wenzel. Catena: A memory-consuming password-scrambling framework. Cryptology ePrint Archive, Report 2013/525, 2013. http://eprint.iacr.org/.

[53] J. Keith Gibson. Discrete logarithm hash function that is collision free and one way. *IEE Proceedings E (Computers and Digital Techniques)*, 138(6):407–410, 1991.

[54] Google Inc. An easy way to pay, purchase, and save – Google Wallet. https://www.google.com/wallet/. Accessed November 2014.

[55] Google Inc. App Engine – Google Cloud Platform. https://cloud.google.com/appengine/. Accessed November 2014.

[56] Google Inc. Compute Engine – Google Cloud Platform. https://cloud.google.com/compute/. Accessed November 2014.

[57] Google Inc. Google Apps for Work – Email, collaboration tools and more. https://www.google.com/work/apps/business/. Accessed November 2014.

[58] Google Inc. Google Cloud Messaging for Android. https://developer.android.com/google/gcm/index.html. Accessed November 2014.

[59] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[60] Helena Handschuh and Bart Preneel. Key-recovery attacks on universal hash function based MAC algorithms. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2008.

[61] Dick Hardt. The OAuth 2.0 authorization framework. RFC 6749, available at https://tools.ietf.org/html/rfc6749, 2012.

[62] Martin E. Hellman. A cryptanalytic time-memory trade-off. *Information Theory, IEEE Transactions on*, 26(4):401–406, 1980.

[63] IEEE 802.1AE. Media access control (MAC) security. http://www.ieee802.org/1/pages/802.1ae.html, 2006. Accessed November 2014.

[64] Google Inc. Google Authenticator project – Two-step verification. `http://code.google.com/p/google-authenticator/`. Accessed November 2014.

[65] Takanori Isobe. A single-key attack on the full GOST block cipher. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2011.

[66] Takanori Isobe and Kyoji Shibutani. Improved all-subkeys recovery attacks on FOX, KATAN and SHACAL-2 block ciphers. To appear at the 21st International Workshop on Fast Software Encryption (FSE 2014).

[67] Takanori Isobe and Kyoji Shibutani. All subkeys recovery attack on block ciphers: Extending meet-in-the-middle approach. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 202–221. Springer, 2012.

[68] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 31–49. Springer, 2012.

[69] Antoine Joux. Authentication failures in NIST version of GCM. NIST Comment, `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/800-38_Series-Drafts/GCM/Joux_comments.pdf`, 2006. Accessed November 2014.

[70] Burt Kaliski. PKCS #5: Password-based cryptography specification version 2.0. RFC 2898, available at `http://www.ietf.org/rfc/rfc2898.txt`, 2000.

[71] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. Chapman & Hall, 2008.

[72] John Kelsey, Bruce Schneier, Chris Hall, and David Wagner. Secure applications of low-entropy keys. In Eiji Okamoto, George I. Davida, and Masahiro Mambo, editors, *Information Security, First International Workshop, ISW '97, Tatsunokuchi, Japan, September 17-19, 1997, Proceedings*, volume 1396 of *Lecture Notes in Computer Science*, pages 121–134. Springer, 1997.

[73] Abdul Nasir Khan, M. L. Mat Kiah, Samee Ullah Khan, and Sajjad Ahmad Madani. Towards secure mobile cloud computing: A survey. *Future Generation Comp. Syst.*, 29(5):1278–1299, 2013.

[74] Dmitry Khovratovich, Gaëtan Leurent, and Christian Rechberger. Narrow-bicliques: Cryptanalysis of full IDEA. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 392–410. Springer, 2012.

[75] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A block cipher for IC-printing. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.

[76] Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, and Sven Verdoolaege. Analysis methods for (alleged) RC4. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings*, volume 1514 of *Lecture Notes in Computer Science*, pages 327–341. Springer, 1998.

[77] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. HMAC: Keyed-hashing for message authentication. RFC 2104, available at `https://tools.ietf.org/html/rfc2104`, 1997.

[78] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In Ivan Damgård, editor, *Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 1990.

[79] Paul Leyland. Factorization of Mersenne numbers, $m_n = 2^n - 1$. `http://www.leyland.vispa.com/numth/factorization/factors/mersenne.txt`, 2008. Accessed November 2014.

[80] Helger Lipmaa, David Wagner, and Phillip Rogaway. Comments to NIST concerning AES modes of operation: CTR-mode encryption. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ctr/ctr-spec.pdf, 2000. Accessed November 2014.

[81] Yiyuan Luo, Qi Chai, Guang Gong, and Xuejia Lai. A lightweight stream cipher WG-7 for RFID encryption and authentication. In *Proceedings of the Global Communications Conference, 2010. GLOBECOM 2010, 6-10 December 2010, Miami, Florida, USA*, pages 1–6. IEEE, 2010.

[82] Tim Mather, Subra Kumaraswamy, and Shahed Latif. *Cloud security and privacy: an enterprise perspective on risks and compliance*. O'Reilly Media, Inc., 2009.

[83] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer Berlin Heidelberg, 1994.

[84] David A. McGrew. Counter mode security: Analysis and recommendations. http://www.mindspring.com/~dmcgrew/ctr-security.pdf, 2002. Accessed November 2014.

[85] David A. McGrew and John Viega. The Galois/Counter Mode of operation (GCM). Submission to NIST Modes of Operation Process, available at http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf, 2004. Accessed November 2014.

[86] David A. McGrew and John Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. In Anne Canteaut and Kapaleeswaran Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer Berlin Heidelberg, 2005.

[87] Peter Mell and Tim Grance. Special Publication 800-145: The NIST definition of cloud computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011. Accessed November 2014.

[88] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[89] The Royal Canadian Mint. The MintChip challenge. http://mintchipchallenge.com/, 2012. Accessed November 2014.

[90] Jezabel Molina-Gil, Pino Caballero-Gil, Cándido Caballero-Gil, and Amparo Fúster-Sabater. Analysis and implementation of the SNOW 3G generator used in 4G/LTE systems. In Álvaro Herrero, Bruno Baruque, Fanny Klett, Ajith Abraham, Václav Snásel, André Carlos Ponce Leon Ferreira de Carvalho, Pablo Garcia Bringas, Ivan Zelinka, Héctor Quintián-Pardo, and Emilio Corchado, editors, *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13 - Salamanca, Spain, September 11th-13th, 2013 Proceedings*, volume 239 of *Advances in Intelligent Systems and Computing*, pages 499–508. Springer, 2013.

[91] National Institute of Standards and Technology. Federal Information Processing Standards Publication (FIPS PUB) 81, DES modes of operation. `http://csrc.nist.gov/publications/fips/fips81/fips81.htm`, 1980. Accessed November 2014.

[92] National Security Agency. Suite B Cryptography. `http://www.nsa.gov/ia/programs/suiteb_cryptography/`, 2005. Accessed November 2014.

[93] Roger M. Needham and David J. Wheeler. TEA extensions. `http://www.movable-type.co.uk/scripts/xtea.pdf`, 1996. Accessed November 2014.

[94] Mozilla Developer Network and individual contributors. Persona protocol overview. `https://developer.mozilla.org/en-US/docs/Mozilla/Persona/Protocol_Overview`. Accessed November 2014.

[95] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33–38, 1994.

[96] OpenID Community. OpenID authentication 2.0 - final. `http://openid.net/specs/openid-authentication-2_0.html`, 2007. Accessed November 2014.

[97] Khaled Ouafi and Serge Vaudenay. Smashing SQUASH-0. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 300–312. Springer, 2009.

[98] Colin Percival. Stronger key derivation via sequential memory-hard functions. BSDCan, 2009.

[99] Thomas Pornin. The MAKWA password hashing function – specifications v1.0. `https://password-hashing.net/submissions/specs/Makwa-v0.pdf`, 2014. Accessed November 2014.

[100] Gordon Procter and Carlos Cid. On weak keys and forgery attacks against polynomial-based MAC schemes. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 287–304. Springer, 2013.

[101] Niels Provos and David Mazières. A future-adaptable password scheme. In *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference, June 6-11, 1999, Monterey, California, USA*, pages 81–91. USENIX, 1999.

[102] Michael O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report, MIT, 1979.

[103] Kui Ren, Cong Wang, and Qian Wang. Security challenges for the public cloud. *IEEE Internet Computing*, 16(1):69–73, 2012.

[104] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 98–107, New York, NY, USA, 2002. ACM.

[105] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.

[106] RSA Inc. RSA SecurID hardware authenticators. `http://www.emc.com/security/rsa-securid/rsa-securid-hardware-authenticators.htm`. Accessed November 2014.

[107] Markku-Juhani Olavi Saarinen. Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 216–225. Springer, 2012.

[108] Salesforce.com, inc. CRM from Salesforce.com – Customer Relationship Management. `https://www.salesforce.com/crm/`. Accessed November 2014.

[109] Tomas Sander. Efficient accumulators without trapdoor extended abstracts. In Vijay Varadharajan and Yi Mu, editors, *Information and Communication Security, Second International Conference, ICICS'99, Sydney, Australia, November 9-11, 1999, Proceedings*, volume 1726 of *Lecture Notes in Computer Science*, pages 252–262. Springer, 1999.

[110] Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2009.

[111] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). In Ross J. Anderson, editor, *Fast Software Encryption, Cambridge Security Workshop, Cambridge, UK, December 9-11, 1993, Proceedings*, volume 809 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 1993.

[112] Bruce Schneier. *Applied cryptography - protocols, algorithms, and source code in C (2. ed.)*. Wiley, 1996.

[113] Ralf Senderek. A discrete logarithm hash function for RSA signatures. `http://senderek.com/SDLH/discrete-logarithm-hash-for-RSA-signatures.ps`, 2003. Accessed November 2014.

[114] Adi Shamir. SQUASH - A new MAC with provable security properties for highly constrained devices such as RFID tags. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 2008.

[115] Victor Shoup. On fast and provably secure message authentication based on universal hashing. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 313–328. Springer, 1996.

[116] Alex Smolen. Login verification on Twitter for iPhone and Android. `https://blog.twitter.com/2013/`

login-verification-on-twitter-for-iphone-and-android, 2013. Accessed November 2014.

[117] Frank Stajano. Pico: No more passwords! In Bruce Christianson, Bruno Crispo, James A. Malcolm, and Frank Stajano, editors, *Security Protocols XIX - 19th International Workshop, Cambridge, UK, March 28-30, 2011, Revised Selected Papers*, volume 7114 of *Lecture Notes in Computer Science*, pages 49–81. Springer, 2011.

[118] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *J. Network and Computer Applications*, 34(1):1–11, 2011.

[119] John Viega and David A. McGrew. The use of Galois/Counter Mode (GCM) in IPsec encapsulating security payload (ESP). RFC 4106, available at http://tools.ietf.org/html/rfc4106.html, 2005.

[120] Lei Wei, Christian Rechberger, Jian Guo, Hongjun Wu, Huaxiong Wang, and San Ling. Improved meet-in-the-middle cryptanalysis of KTANTAN (poster). In Udaya Parampalli and Philip Hawkes, editors, *Information Security and Privacy - 16th Australasian Conference, ACISP 2011, Melbourne, Australia, July 11-13, 2011. Proceedings*, volume 6812 of *Lecture Notes in Computer Science*, pages 433–438. Springer, 2011.

[121] Bo Zhu, Kefei Chen, and Xuejia Lai. Bitwise higher order differential cryptanalysis. In Liqun Chen and Moti Yung, editors, *Trusted Systems, First International Conference, INTRUST 2009, Beijing, China, December 17-19, 2009. Revised Selected Papers*, volume 6163 of *Lecture Notes in Computer Science*, pages 250–262. Springer, 2009.

[122] Bo Zhu and Xinxin Fan. EasyChip – Submission to the MintChip Challenge. http://mintchipchallenge.com/submissions/9469-easychip, 2012. Accessed November 2014.

[123] Bo Zhu, Xinxin Fan, and Guang Gong. Pleco and Plectron – Two provably secure password hashing algorithms (poster). To appear at the Fifth ACM Conference on Data and Application Security and Privacy (ACM CODASPY 2015). The full version has been submitted to the IEEE Transactions on Dependable and Secure Computing.

[124] Bo Zhu, Xinxin Fan, and Guang Gong. Loxin – A solution to password-less universal login. In *2014 Proceedings IEEE INFOCOM Workshops, Toronto, ON, Canada, April 27 - May 2, 2014*, pages 488–493. IEEE, 2014.

[125] Bo Zhu, Xinxin Fan, and Guang Gong. Pleco and Plectron – Two provably secure password hashing algorithms. Cryptology ePrint Archive, Report 2014/655, 2014. http://eprint.iacr.org/.

[126] Bo Zhu and Guang Gong. Multidimensional meet-in-the-middle attack and its applications to KATAN32/48/64. *Cryptography and Communications*, 6(4):313–333, 2014.

[127] Bo Zhu, Guang Gong, Xuejia Lai, and Kefei Chen. Another view on cube attack, cube tester, AIDA and higher order differential cryptanalysis. CACR Technical Report 2012-01, available at http://cacr.uwaterloo.ca/techreports/2012/cacr2012-01.pdf, 2012.

[128] Bo Zhu, Yin Tan, and Guang Gong. Revisiting MAC forgeries, weak keys and provable security of Galois/Counter Mode of operation. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *Cryptology and Network Security - 12th International Conference, CANS 2013, Paraty, Brazil, November 20-22. 2013. Proceedings*, volume 8257 of *Lecture Notes in Computer Science*, pages 20–38. Springer, 2013.

[129] Larry Zhu and Brian Tung. Public key cryptography for initial authentication in Kerberos (PKINIT). RFC 4556, available at http://www.ietf.org/rfc/rfc4556.txt, 2006.